ROBERT E. SMITH Ph.D.
and
DORA E. JOHNSON

FORTRAN

AUTOTESTER

JOHN WILEY & SONS, INC.
NEW YORK • LONDON • SYDNEY

FORTRAN AUTOTESTER

ROBERT E. SMITH Ph.D.
and
DORA E. JOHNSON
Control Data Corporation
Minneapolis, Minnesota

A self-training course designed to emancipate the scientist and engineer from the need for the professional programmer

John Wiley & Sons, Inc.,
New York • London • Sydney

SEVENTH PRINTING, JANUARY, 1967

Printed in the United States of America

PREFACE

From time to time scientific and engineering personnel find themselves in situations where there is a need to know how to "program" their problems for the modern electronic digital computer.

To learn machine language or assembly language programming requires too much time; there are too many petty rules, and numerous exceptions are involved. Unless the scientist can devote full time to the effort, he looks down the "long road ahead" and is discouraged from even trying to start.

This short course is an attempt to give to the scientist and engineer, sufficient skill, in a minimum of time, to enable him to efficiently program his own problems. To accomplish this objective, the course contains the following characteristics:

- The FORTRAN language is used exclusively!
- Non essential concepts have been eliminated!
- "Learning Machine" techniques are used--in that continued lesson successes depend upon learning previous concepts!
- Every effort (through "humor", etc) is made to keep you awake!

GETTING STARTED?

. . . . is so easy?

SETTLE DOWN?

. . . . in anything?

SCRATCH PAPER?

. . . . yes, have some at hand!

FORTRAN MANUALS?

. . . . not needed, but available!

IS THAT ALL?

. . . . yes, go to Card 1.

"Her name is Constance", said the man, "my ever faithful and unchanging wife". In a similar vein, the mathematician likes to call unchanging magnitudes, CONSTANTS. How many constants would you say there are among the following?

a) 1062

b) 3/4

c) 1062.

d) .52E2 ("E2" indicates 2 is the exponent of 10)

If your answer is 4, go to 3

If your answer is less than 4, go to 2

You probably thought the fourth example was not a CONSTANT because of the letter "E". This is a good guess, but it is too cautious! The "E" is just a symbol to indicate "exponent of 10". Some constants are more easily expressed this way. For example, the constant:

62,000,000,000,000,000,000,000,000,000,000 = .62E32

Go To 3

Now that your concept of CONSTANTS is O. K., do you see a basic format difference in the constants, 1062 and 1062.? Yes, there is a decimal point expressed in one. Fortran makes use of two types of constants, FIXED POINT CONSTANTS and FLOATING POINT CONSTANTS.

Floating point format makes it possible to do arithmetic (add, subtract, etc.) and have the computer automatically position the decimal point, thus making it unnecessary for you to continually scale and rescale as the problem progresses. The compiler must know what type of CONSTANT you are thinking about. Therefore:

IF THE DECIMAL POINT IS EXPRESSED, YOU ARE USING FLOATING POINT CONSTANTS.

IF THERE IS NO DECIMAL POINT EXPRESSED, YOU ARE USING FIXED POINT CONSTANTS.

Go To 4

A plus (+) or minus (-) sign preceding a constant indicates a <u>positive</u> or <u>negative</u> constant. A plus (+) need not be written with positive constants, but the minus (-) <u>must</u> <u>be</u> <u>written</u> with negative constants. In summary then, if you use:

3.2	-6	35
.02	3	23
0.	0	35
-3.	+2	
.6231E-7	61235	
etc.	etc.	
you are thinking about FLOATING POINT CONSTANTS!	you are thinking about FIXED POINT CONSTANTS!	you are thinking about "LIVE" CONSTANCES!

Go To 5

In addition to constants, you will make use of variables. VARIABLES are quantities that may assume a succession of changing magnitudes. Whereas the representation of each constant is the constant itself, each variable must be represented by a combination of letters and digits.

FIXED POINT VARIABLES only represent integers or whole numbers--expressed without a decimal point

FLOATING POINT VARIABLES represent fractional, mixed, or whole numbers with a decimal point expressed.

The following four rules apply for variables:

(1) The <u>first character</u> of the variable name must be alphabetic.

(2) A first character of I, J, K, L, M, or N indicates a FIXED POINT VARIABLE; any other first letter indicates a FLOATING POINT VARIABLE.

(3) Each variable name can not contain more than 7 characters.

(4) No special characters (such as, +, -, etc.) are permitted.

Go To 6

Examples of FIXED POINT VARIABLE names are: I, NEMO, M5, and LEFTONE.

How many of the following FIXED POINT VARIABLE names are not permitted in Fortran?

INCREASE	MAP	KONSTANT
N3+J	I234567T	430126
I5	JAX	LEFT3

If your answer is 3 or 4, re-read 5, and then repeat 6.

If your answer is 5, GO TO 7.

If your answer is 0, 1, 2, 6, 7, 8, or 9, GO TAKE A SHOWER and repeat 5.

Examples of FLOATING POINT VARIABLE names are:

A32 CREDIT DEBIT Y OK GAMMA FORTRAN

How many of the following FLOATING POINT VARIABLE names are <u>not</u> permitted?

NO32.6	PI	ALPH-BETA
XYZ	E	ALP-BET
3.2	ALPHA	Z72

If your answer is not 4, review 5 then repeat 7.
If your answer is 4, go to 8.

The word "FORTRAN" is a contraction of "<u>For</u>mula <u>Tran</u>slation". A Fortran arithmetic statement closely resembles a conventional mathematical formula. An important conceptual exception is the equal, =, sign. In Fortran, "=" means "is to be replaced by" rather than "is equivalent to". Thus, in Fortran, the statement:

Y = A3 + BIG

means that the sum of the floating point variables, A3 and BIG, is to replace the present value of the floating point variable, Y.

Another important exception is that the left hand member of a Fortran statement must be a SINGLE VARIABLE NAME---it can <u>not</u> be an expression.

Thus, I2 = 3 + J5 is permitted, but I2 + J5 = 3 is not.

Go To 9

The sum of 20 terms of the arithmetic series, 3 + 5 + 7 ----- 41, is 440. This is easy, if one remembers the rule:

$$\text{Sum} = \frac{\text{Total Number of Terms}}{2}\left(\begin{matrix}\text{First}\\\text{Term}\end{matrix} + \begin{matrix}\text{Last}\\\text{Term}\end{matrix}\right)$$

Let us turn this rule into a formula by use of the following letters:

$$S = \frac{TN}{2}(A + B)$$

In Fortran, the asterisk, *, is used as the multiplication symbol and the slash, /, as the division symbol. A double astrisk, **, indicates exponentiation. Otherwise, other symbols including parentheses are conventional. With this short briefing, can you write a Fortran statement for the above formula? Try it---on a separate sheet of paper, then

Go To 10

Congratulations if you have one of the following! Probably the second or third best fits the "left to right" operational philosophy.

```
S = (A + B) * TN/2.
S = (TN/2.) * (A + B)
S = .5 * TN * (A + B)
S = (TN * (A + B))/2.
S = TN * (A + B)/2.
```

Note the importance of the decimal point after the 2 and before 5. Constants must be floating point if the other variables of the right hand member are floating point. (This will be explained later.)

FORTRAN ORDER OF OPERATIONS PHILOSOPHY

Quantities within parentheses are computed first and the resultant value is then considered in the simplified expression. All operations are performed from left to right in the following order:

(1) Exponentiation (**)

(2) Multiplication (*) and Division (/)

(3) Addition (+) and Subtration (-)

Go To 11

Let us follow through an example using the correct FORTRAN "order of operations" philosophy.

Example: A = B + C/D ** E * (F + 2.0)

The calculation would proceed as follows:

Step 1	Add F to 2.0
Step 2	Raise D to power E
Step 3	Divide C by the result of step 2
Step 4	Multiply the result of step 3 by the result of step 1
Step 5	Add B to the result of step 4
Step 6	Replace A with the result of step 5

Go To 12

A student programmer recently wrote the following statements for the example below. Determine which of the statements are correct.

Example: Add 3 to a variable, J, and square the sum. Multiply the product by 5 and divide by 12. Let the final result equal K.

STATEMENTS

(1) K = 5 * (J + 3) ** 2/12

(2) K = 5 * J + 3 ** 2/12

(3) 5 * (J + 3) ** 2/12 = K

(4) K = (J + 3) * 5 ** 2/12

(5) K = (J + 3) ** 2 * 5/12

If your answer is 1 and/or 5, go to 14

If your answer is 2, 3, or 4, go to 13

The first Law of Revision:

Information necessitating a change in the program will be conveyed to the programmer after -- and only after -- the program has been debugged!

Statement (2), K = 5 * J + 3 ** 2/12, is incorrect. It is equivalent to: $K = 5J + \frac{3^2}{12}$

Statement (3), 5 * (J + 3) ** 2/12 = K, is correct mathematically. However, in Fortran the side to the left of the equal sign can <u>only</u> be a single variable. Thus if the sides were reversed, this answer would be correct.

Statement (4), K = (J + 3) * 5 ** 2/12, is incorrect. It is equivalent to:

$$K = \frac{(J + 3)\ 5^2}{12}$$

Why not return to 12 and try that example once more.

The two examples just completed involved the following statements:

(1) A = B + C/D ** E * (F + 2.0)   (2) K = 5 * (J + 3) ** 2/12

In the first statement both members are in "floating mode", since they contain only FLOATING POINT VARIABLES and FLOATING POINT CONSTANTS. In the second, both are in "fixed mode", since they contain only FIXED POINT VARIABLES and FIXED POINT CONSTANTS.

The next question becomes very important----CAN THE LEFT SIDE BE OF A DIFFERENT MODE THAN THE RIGHT SIDE? The answer is, Yes, <u>but only under the following restrictions</u>:

(1) The left side (FIXED or FLOATING) must be a single variable.

(2) If the modes of the sides differ, calculations are done using the mode of the right side and then put into the mode of the left. Thus in F = I + 3, the FIXED POINT, I + 3, is FLOATED and replaces F. However, in I = 1.5 + A; when the FLOATING POINT sum, 1.5 + A, is FIXED, the fractional part of the sum is lost. Thus, if I = 1.5 + 2.2; I is replaced by 3.

Go To 15

NOW, CAN ONE MIX MODES WITHIN THE RIGHT SIDE OF A STATEMENT?

Yes, but again only under certain cases. There are 3 such cases:

(1) EXPONENTS

(2) SUBSCRIPTS

(3) ARGUMENTS OF SUBROUTINES

The last two are described later. The first is easily learned. It seems logical to be permitted to use an integral FIXED POINT exponent with a FLOATING base. In fact, a negative base requires a FIXED POINT exponent.

Examples:

```
B = (-X)**3
Y = TP + 3.2 - H**7
```

The integer, 7, is an exponent and is permitted in FIXED mode even though the right side is FLOATING. However, one is not restricted from using FLOATING POINT exponents if desired. For example:

```
Y = TP + 3.2 - H**7.
Y = TP + 3.2 - H**3.2
```

Go To 16

Several Goodies have now been presented to you. See how many you sampled, by writing the answers to the following?

(1) Mathematically 716. and 716 are equivalent. In Fortran, is there any difference?

(2) What constant does .0126 E + 4 represent?

(3) Three variables are IKE, MIKE, and TOM. Which one represents fractional quantities?

(4) Complete the limerick:

Yes, dear Variable, the name is "Pettycur"
It fits this data like "good" fits heaven
But, alas I can't use the moniker
Since the characters number __________.

(5) White <u>can</u> <u>not</u> equal black in color photography. Can WHITE = BLACK in Fortran?

(6) List the arithmetic symbols used in Fortran.

(7) Given: 7X + 12 = Y, write a Fortran formula which would solve this equation for X.

(8) Write the following as a Fortran statement and find the value of I.

$$I = 2\,(37 + 3) - \frac{2(4 + 1)^2\,(1 + 1)}{5} + 1$$

(9) Is the statement J = 3.2 + 5.4 - 6.123 permitted in Fortran? If so, what will be the final value of J?

Go To 17

Dear Godfrey, here are the TIPSTERS you need. I hope your's are correct: ---

(1) There is a difference. "716" is a FIXED POINT CONSTANT, but "716." is a FLOATING POINT CONSTANT.

(2) .0126E + 4 is another way of writing the constant 126.

(3) IKE and MIKE represent integers--variable names starting with I, J, K, L, M, and N represent FIXED POINT quantities. TOM represents a fractional quantity--it is a FLOATING POINT variable, since it does not begin with letters I, J, K, L, M, or N.

(4) "Since the characters number <u>more</u> <u>than</u> <u>seven</u>!

(5) Yes. In Fortran WHITE = BLACK means that the value of variable, WHITE, is replaced by the value of variable, BLACK.

(6)

+ add	* multiply	= equal
- subtract	/ divide	( ) grouping
	** exponentiation	

(7) X = (Y - 12.)/7.

(8) I = (37 + 3) * 2 - (2*(4 + 1)**2*(1 + 1))/5 + 1   and   I = 61

(9) Yes. The right member will be calculated in FLOATING POINT (2.477), and then FIXED variable, J, will be set equal to 2 (FIXED value of 2.477).

Go To 18

One often wants to represent a whole ARRAY of variables by a single variable name with individual terms of the array being designated by using subscripts with the array name. Provision is made in Fortran for one, two, or three dimensional arrays. The array name is followed by the subscripts separated by commas, and enclosed by parentheses. Thus A_1 is defined as A(1); $A_{1,2}$ is defined as A(1, 2); and $A_{1,2,3}$ as A(1,2,3).

The subscript must be either a FIXED POINT constant, FIXED POINT variable, or FIXED POINT expression. For example; A(1), A(J), A(J + 3).

Note that a FIXED POINT SUBSCRIPT can be used in a FLOATING ARRAY. For example, BETA(4). Here is the second permissable use of mixed modes within the right side of an arithmetic statement. Thus, a FIXED POINT subscript may appear with a FLOATING POINT variable name.

Note that whereas the subscript is always FIXED, the array name itself can be FIXED or FLOATING. Thus I(1) is permitted as well as A(1); JET(2, 3) as well as BET(2, 3); etc.

Go To 19

The Compiler must know how many locations to reserve for arrays. This information is given to the Compiler by a DIMENSION statement which must precede the first reference to the arrays. One DIMENSION statement can indicate the range of more than one array. For example:

```
DIMENSION A(200), IOTA(50, 50), PI(3, 3, 3,)
```

The above DIMENSION statement provides for a one dimensional array, A, of 200 terms; a two dimensional array, IOTA, of 50 by 50 (2500) terms; and a three dimensional array, PI, of 3 by 3 by 3 (27) terms.

Do you see what is required in a DIMENSION statement? The word, DIMENSION, is followed by the array name, followed by parentheses enclosing the dimensional maximums for the subscript,which are separated by commas. Commas also separate one array from another in the statement list.

Go To 20

Alphonse used a 20 by 30 matrix (array) to represent a system of equations. He wanted to add to the term located in the 5th row and the 23rd column. Alphonse wrote the following statements to indicate the range of the array and to increase the specified term of the array.

```
DIMENSION ALPHA (20, 30)
ALPHA (5, 23) = ALPHA (5, 23) + 3.
```

Did Alphonse GOOF?

If you think he did, go to 21
If you agree with Alphonse, go to 22

No, Alphonse did not goof---check the problem again, and then study the following example---for more practice!

Alphonse also used an array, DATA, of 2000 terms. Various terms of this array were processed, depending upon the value of N. Thus if N = 3, then $DATA_3$ was processed; if N = 7, $DATA_7$, was processed, etc.

Let us assume that during part of the program, Alphonse desires to multiply the third term <u>following</u> the processed term, by 5 and put the result at X. Thus if N = 3, he wants to put 5($DATA_6$) at X. He writes the following statements:

```
DIMENSION DATA (2000)
X = DATA (N + 3) * 5.
```

Note the decimal point after the constant, 5, since "DATA" indicates FLOATING POINT.

Go To 22

Alphonse later set up the following statements for another problem:

```
DIMENSION T(9), K(8)
I = 1
J = 0
T(I + J) = 3.0
J = J + 1
T( I + J)= 4.2
T(I + J + 7) = T(I + J - 1) + 1.2
K(7) = T(I + J)
```

The above statements would result in specific actions. On a separate piece of paper mark the following as TRUE or FALSE.

(1) The DIMENSION statement indicates a 9 term array (T) and an 8 term array (K).
(2) The value of T_1 is replaced by FLOATING POINT 3
(3) The value of T_2 is replaced by a FLOATING POINT 4.2
(4) The value of T_9 is replaced by a FLOATING POINT 4.2
(5) The value of K_7 is replaced by a FLOATING POINT 4.2

If you think <u>all</u> 5 were TRUE, go to 23
If you think at least 4 were TRUE, go to 24
If you think at least 3 were TRUE, go to 25

The 5th one was FALSE! The value of K_7 must be replaced by T_2 as claimed, and $T_2 = 4.2$ is in FLOATING POINT. But K_7 must be FIXED POINT since K is one of the six FIXED POINT letter indicators (I, J, K, L, M, N). Therefore, $K_7 = T_2$ is a statement in which the mode of the left side of the statement differs from the mode of the right side. The mode of the <u>left</u> side determines the final mode; so in this case, the final mode must be FIXED POINT, since "K($_7$) is the left variable. The correct answer, then, is:

$$K_7 = 4$$

The fractional part of the result has been lost!

Go To 26

You are on the beam, and to show you how proud we are of your good work, we present the following story:

Years ago in India, an old tea taster, Sanskrit, always confounded the experts by unfailingly being able to differentiate between BOMBAY, CALCUTTA, and KILMNJ teas. In those days it was common for unscrupulous tea makers to try to push KILMNJ tea on unsuspecting merchants. But old Sanskrit always could tell the difference.

In spite of all efforts to discover his secret, Sanskrit carried it to his grave. On his death bed he is supposed to have muttered something like the following, "They FLOAT, but KILMNJ is FIXED".

Years later, a far removed American descendent of Sanskrit remembered these words. This distant cousin in writing a Fortran Compiler, and faced with the problem of differentiating between FIXED and FLOATING point variables, noticed that "KILMNJ" when re-arranged had the alphabetic order of I-J-K-L-M-N.

To this day, in Fortran, FIXED POINT variables begin with I, J, K, L, M, or N. Otherwise, they FLOAT.

What happened? Do you feel well? Let's go over the statements.

DIMENSION T(9), K(8)	Reserves 9 locations for FLOATING POINT array, T, and 8 locations for FIXED POINT array, K.
I = 1	1 goes to I
J = 0	0 goes to J
T(I + J) = 3.0	T(I + J) = T(1 + 0) = T_1 is replaced by 3.0
J = J + 1	J + 1 = 0 + 1 = 1, goes to J
T(I + J) = 4.2	T(I + J) = T(1 + 1) = T_2 is replaced by 4.2
$\left\{\begin{array}{l}\text{T(I + J + 7) =}\\ \text{T(I + J - 1) + 1.2}\end{array}\right.$	T(I + J + 7) = T(1 + 1 + 7) = T_9 is replaced by T(I + J - 1) + 1.2 = T(1 + 1 - 1) + 1.2 = T_1 + 1.2 = 3.0 + 1.2 = 4.2
K(7) = T(I + J)	T(I + J) = T(1 + 1) = T_2 = 4.2 is to go to K_7, But K_7 is FIXED. Thus the fractional part is lost, and 4 goes to K_7.

Let's start again,

Go To 22

So far you have been doing very well but you can't afford to become over-confident. Let's stop for a couple of minutes and try another set of exercises before moving on. Write the answers to the following:

A. Write one statement which will reserve enough space for W_1, W_2, --- W_{350}; and also for a 50 by 50 matrix called MONO.

B. A formula for CHI SQUARE is: Write this in Fortran.

$$CHI = \frac{T\left[(A)\ (D) - (B)\ (C) + \frac{T}{2}\right]^2}{(A + B)\ (A + D)\ (B + C)\ (C + D)}$$

C. Given: TETRA = P3 ** 2 + 5.2 + V5 Since everything in the right member is FLOATING POINT, the constant 2 must be 2.0 or 2.. Defend or attack this conclusion!

D. May a subscript be a FIXED POINT constant? If yes, give an example.
May a subscript be a FIXED POINT variable? If yes, give an example.
May a subscript be a FIXED POINT expression? If yes, give an example.
May a subscript be a FLOATING POINT constant, variable, or expression?
May a subscripted variable name be FIXED or FLOATING? If yes, give an example.

Go To 27

CHECK YOUR ANSWERS!

A. DIMENSION W(350), MONO (50, 50)

B. CHI = T * (A * D - B * C + T/2.) ** 2/((A + B) * (A + D) * (B + C) * (C + D))

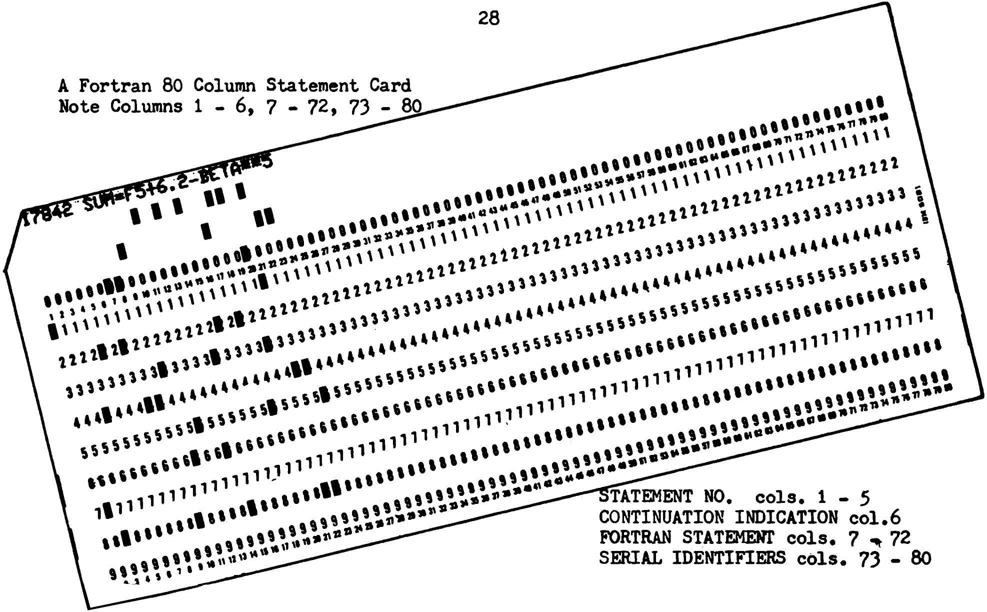
C. It is <u>not</u> necessary to give the decimal point! Nor is it wrong! Keep in mind, that an <u>exponent</u> <u>may</u> be a FIXED POINT quantity even though the base may be FLOATING POINT. This is <u>one</u> of the cases where a FIXED quantity is allowed to be <u>mixed</u> with FLOATING variables. A second case is SUBSCRIPTS. The third case, which you will get later, involves the uses of ARGUMENTS of functions.

D. Yes, a subscript can be a FIXED POINT constant; ex. U(3)
Yes, a subscript can be a FIXED POINT variable; ex. U(NEND)
Yes, a subscript can be a FIXED POINT expression; ex. U(I + J - 3)
No, a subscript can <u>not</u> be a FLOATING POINT constant, variable, or expression.
Yes, the subscripted variable name can be FIXED or FLOATING; ex. J(3) or T(3)

If you had significant differences from the above, REVIEW 18 and 19, then

Go To 28

A Fortran 80 Column Statement Card
Note Columns 1 - 6, 7 - 72, 73 - 80



STATEMENT NO. cols. 1 - 5
CONTINUATION INDICATION col.6
FORTRAN STATEMENT cols. 7 - 72
SERIAL IDENTIFIERS cols. 73 - 80

You just saw a FORTRAN STATEMENT CARD. Did you really GRASP the important facts presented? Wait! Don't look at it! Try these questions first!

In what column did the Fortran statement start?

In what column must the Fortran statement end?

In what columns are the Statement Numbers given?

If there is a CONTINUATION indication---what column should it be in?

Where are Serial Identifications made?

Now, look at 28 again!

If you feel you did O.K., then

Go To 30

Column 7 is the reference column to remember, when writing Fortran programs! All statements or continued statements begin there! Starting at column 7 statements progress through column 72 --- ONE CHARACTER PER COLUMN. Remember, only ONE statement is allowed on the <u>same card</u>.

If a statement extends beyond column 72, then a CONTINUATION card is required. Column 6 is used when CONTINUATION cards become necessary. A <u>blank</u> in column 6 indicates <u>no</u> CONTINUATION; NINE digits 1, 2, 3, --- 9 indicate corresponding CONTINUATIONS. Thus, if <u>one</u> statement requires 3 cards; the first card has a blank in column 6; the second card has a 1 in column 6; and the third card, a 2 in column 6. Remember, each continued statement also starts in column 7 -- following the continuation digit in column 6.

Statement numbers are given in columns 1 - 5. All statements need not have a number-- only those which are later referenced in the program by other statements.

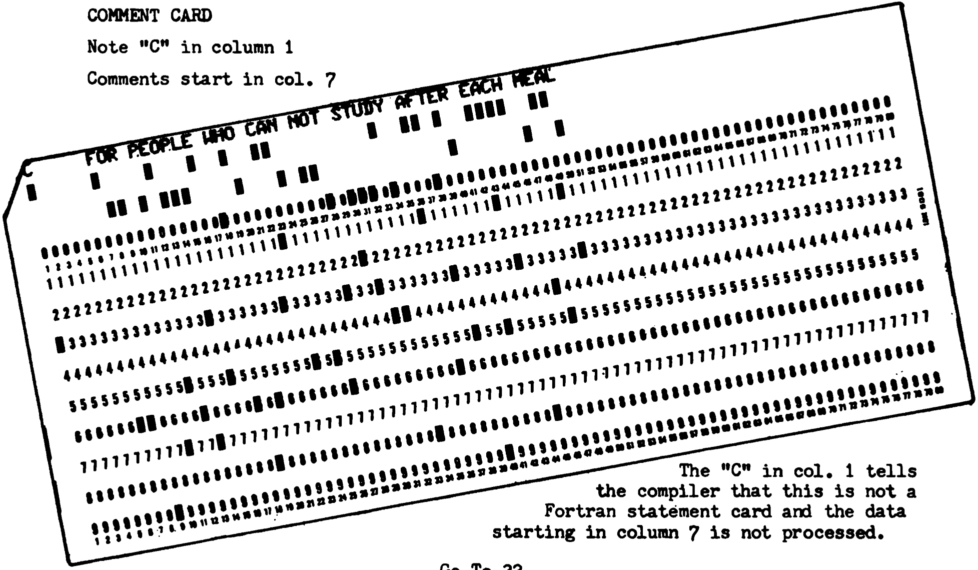
Columns 73 - 80 need not be used as the compiler does not process these. However, they may be used for serializing the cards, or providing other information.

Go To 31

COMMENT CARD

Note "C" in column 1

Comments start in col. 7



The "C" in col. 1 tells the compiler that this is not a Fortran statement card and the data starting in column 7 is not processed.

Go To 32

You have seen the 2 types of cards which are source programs prepared for input to the compiler--STATEMENT cards and COMMENTS cards. A key punch operator will probably punch these cards for you. However, it is important for you to know card format so that you can write programs in correct form for the operator to punch--otherwise, they are liable to punch you! Your programming will probably be done on printed forms similar to 33. (These FORTRAN CODING forms are available from the Publications Dept.)

33 portrays the requirements of card format you have just reviewed. Study this example carefully, and note the column sub-divisions.

P. S. We are assuming that you have "picked up" the <u>correct</u> concept that all constants used in Fortran--including Statement Numbers--are our "good old" <u>decimal</u> <u>numbers</u>--the Compiler takes care of all conversion's for you.

Go To 33

1604 FORTRAN CODING FORM

NAME.

PROGRAM	Doomed	PAGE	1
ROUTINE	Shot to the Moon	DATE	6/7/99

TYPE 1	STATEMENT NUMBER 2 – 5	CONT. 6	FORTRAN STATEMENT 7 – 72	SERIAL NUMBER 73 – 80
			PROGRAM DOOMED	124
C			A PROGRAM TO END ALL PROGRAMS	125
	6		DEATH = SHOT3 + 5.2	126
			CHI = T * (A*B - C*D + T/2.)**2/(	127
		1	(A + B) * (C + D) * (A + D) * (B + C))	128
			GO TO 6	129

Serial 124 is a PROGRAM NAME CARD -- required for identification
Serial 125 is a COMMENT CARD (Indication in Column 1)
Serial 128 is a CONTINUATION CARD (Indication in Column 6)
Serial 129 is a CONTROL STATEMENT CARD (To be discussed later)

Go To 34

How well do you know Card Format? Try matching the following numbers and letters on a work sheet.

1. Serialized Card numbers
2. Column 6
3. Normal Statement Card
4. "C" in column 1
5. Card spaces are ignored by the compiler
6. Can a statement of more than 660 characters be used?
7. Can not be put on one card.
8. A Fortran statement should start in column 7.
9. Statement numbers need not be given in numerical order.

A. True

B. Punch in columns 1 and 6

C. Col. 1 is blank or numeric

D. Yes

E. Two Fortran Statements

F. False

G. Comments Card

H. Continuation indication column

I. Columns 73 - 80

J. No

Go To 35

The correct matching should be:

(1I, 2H, 3C, 4G, 5A or 5D, 6J, 7E, 8A or 8D, 9A or 9D)

If you missed a couple, blame us since the test contained a few "sneaky items".

Are spaces ignored by the compiler? Yes. We hoped you would assume this to be true. Note, in example of 33, we used some spaces.

Is 660 the maximum number of characters permitted? Yes. There may be 9 CONTINUATION CARDS plus the original card--making 10 cards total. Each card uses columns 7 through 72 (66 characters) for a grand total of 10(66) = 660. Note, the 660 means 660 columns <u>including</u> any spaces that are involved within the statement.

Also statement numbers need not be in numerical order--we had not mentioned this previously---but now you know it!

Go To 36

The name of a program is "ANIMAL". Ten variables: A, ALPHA, B, C, D, HIPPO, LYNX, BEAR, I, and J are involved.

HIPPO, LYNX, and BEAR are subscripted. HIPPO is a 60 by 60 matrix, LYNX is an array of 12 terms, and BEAR is a 16 by 16 by 16 matrix. The program is identified as the "ANIMAL PROGRAM". The following formula is used in the program and is referred to later as statement 22.

$$HIPPO_{i,\,50} = LYNX_{11} + (I + J)^3$$

Also the following formula is used and is subdivided into CONTINUATION statements. For this example, assume the continuations start with each slash (division) shown.

$$BEAR_I = A(C + D)^2 \;/\; B + (B)\;(C + D)^3/\;(ALPHA + 3.2)$$

Program the above example, using correct card format, and indicating the card column sub-divisions.

Go To 37

1	2 STATEMENT NUMBER COLUMNS 5	CONT. 6	7 FORTRAN STATEMENT COLUMNS 72	73 80
			PROGRAM ANIMAL	
C			ANIMAL PROGRAM	
			DIMENSION HIPPO (60, 60), LYNX(12), BEAR(16, 16, 16)	
	22		HIPPO(1,50) = LYNX(11) + (I + J) ** 3	
			BEAR (I) = A * (C + D) ** 2	
		1	/ B + B * (C + D) ** 3	
		2	/ (ALPHA + 3.2)	

Go To 38

A computer program is similar to the recipe for "making a cake" in that certain cake ingredients are listed followed by a set of directions or order of events to put these together.

Up to this point, we hope you have learned the Fortran ingredients: symbols, card format, rules of constants, variables, etc. We now need to learn certain CONTROL statements which will give direction to the normal sequential order of events of a written program.

Although Fortran includes a few more CONTROL statements than will be presented in the course, the following seven will be sufficient to provide efficient control for any Fortran program.

GO TO (unconditional)	CONTINUE
GO TO (computed)	PAUSE
IF	STOP
DO	

Go To 39

The first CONTROL statement is the unconditional "GO TO". This statement has already been used at the bottom of each page -- where the number expressed is a page number instead of a statement number. Thus, the correct form is, GO TO n, where n is a statement number. An example is: GO TO 372.

The second CONTROL statement is similar. The computed "GO TO" must include a list of statement numbers, separated by commas and enclosed by parentheses. This list is followed by a comma and a non-subscripted fixed point variable, such as I. The value of I then determines which branch to take.

For example GO TO (100, 72, 136), J

if J = 1, control will go to statement 100

if J = 2, control will go to statement 72

if J = 3, control will go to statement 136

if J = 0, 4, or larger (in this example), an error halts the program.

Go To 40

Although the first GO TO is self evident, the second needs some practice. Below are some further examples of computed GO TO statements.

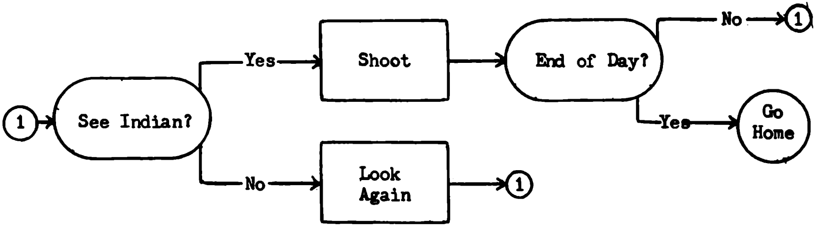
```
GO TO (45, 129, 5, 5, 5),JAKE
If JAKE = 1, GO TO Statement Number 45
If JAKE = 2, GO TO Statement Number 129
If JAKE = 3, 4 or 5, GO TO Statement Number 5

GO TO (7, 18),M
IF M is 1, GO TO Statement Number 7
IF M is 2, GO TO Statement Number 18
```

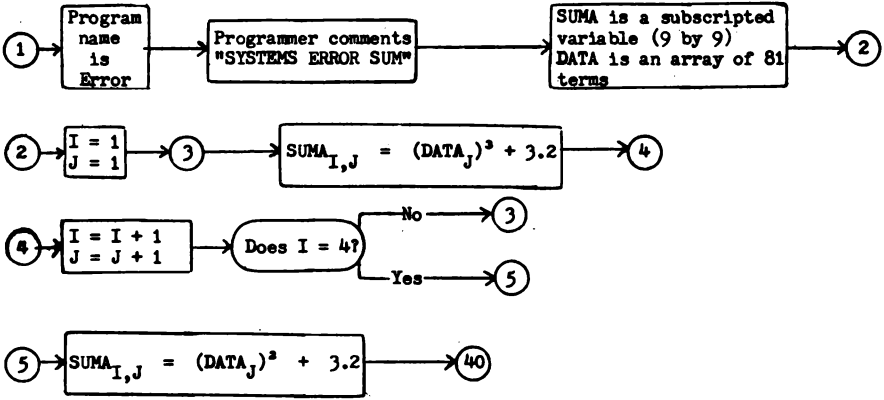
You are to be guided by the GO TO Statement below, assuming NN is 3.

```
Go To (40, 40, 41),NN
```

R. U. HESITANT: born 1782, before advent of modern computer; known as "father of the flow chart", planned events of each day with a diagram similar to following:



Given the following flow chart of part of a program; write Fortran statements which will do everything shown in flow chart.



Go To 42

Your Fortran program from the flow chart should be the following:

```
Columns
1 ---5 6 7
        PROGRAM ERROR
C       SYSTEMS ERROR SUM
        DIMENSION SUMA (9, 9), DATA (81)
        I = 1
        J = 1
    3   SUMA (I, J) = DATA (J) ** 3 + 3.2
    4   I = I + 1
        J = J + 1
        GO TO (3, 3, 3, 5),I
    5   SUMA (I, J) = DATA (J) ** 2 + 3.2
        GO TO 40
```

If you missed some, REVIEW 38-40 and try 41 again

Otherwise:

Go To 43

The next CONTROL statement is the "IF" statement.

IF (X - Y) 20, 30, 15

This example means, "If the value of X - Y is less than zero, go to statement 20; if equal to zero, go to statement 30; and if greater than zero, go to statement 15."

IF (X) 20, 30, 15

This example is identical except the value of "X" is now examined as to whether it is less than, equal to, or more than zero.

IF (J + 2) 15, 15, 20

In this example, "J + 2" is examined. If J + 2 is less than or equal to zero, control goes to statement 15; if more than zero, to statement 20.

Go To 44

Summarizing the IF statement then the format is:

IF	(FIXED or FLOATING VARIABLE or FIXED or FLOATING EXPRESSION)	Less than zero branch Statement number ,	Equal to zero branch Statement number ,	Greater than zero branch Statement number

Following the parentheses are 3 statement numbers separated by commas. These are the paths or branches. If the quantity in parentheses is <u>less</u> than zero control goes to the first statement number, if equal to zero, control goes to the second statement number; and if greater than zero, control goes to the third statement number. <u>All 3 statement numbers must appear</u>.

Another example:

```
IF (BETA(I))  10, 20, 10
```

Go To 45

Write IF statements for the following three examples.

(1) The variable JET is usually negative; However, it can become zero. If JET ever becomes greater than zero, an error results. As long as JET is negative, the branch is to statement 11; if zero, it is to statement 22. The error stop is at statement 5.

(2) If the expression "DATA + Y - .5E2" is zero, the branch is to statement number 6; otherwise, the branch is to statement number 100.

(3) If W^e is negative the control goes to an error exit at statement 42; if zero, control goes to statement 5; otherwise to statement 15.

Go To 46

Answers to the 3 "IF" examples are:

(1) IF (JET) 11, 22, 5

(2) IF (DATA + Y - .5E2) 100, 6, 100

(3) IF (W ** 2) 42, 5, 15

Did that constant in example 2 look strange? Remember, .5E2 is a FLOATING POINT constant equivalent to $.5(10)^2 = 50$.

Note in second example, the IF statement becomes a two way branch. Also note the "pseudo example" given below, is a one way branch or an unconditional GO TO.

IF (YOU MISSED ANY OF ABOVE) 43, 43, 43

Otherwise,

Go To 47

"How do I get to Casey street?", I asked the cop.

"Go to 15th, and check the SIGNAL there. If red, go right to 50th, if yellow turn left to 34th, and if green, go straight ahead to 11th."

"But I'm color blind", I protested, "how can I tell if the signal is red, yellow or green?"

"It's easy", said the cop. "When you get there check the amount of money you have in your left coat pocket. If none, the light is yellow, if at least one cent, the light is green, if you find a hole in your pocket, the light is red." "Very simple", he concluded boastfully, "if you know FORTRAN".

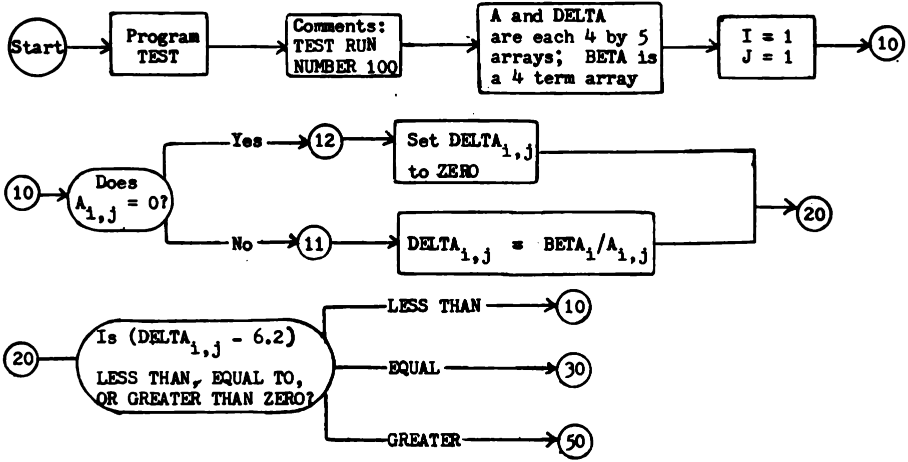
I thought for a minute, wrote the following on a sheet of paper, and handed it to the cop, with the question, "Is this what you mean"?

```
      GO TO 15
   15 IF (MONEY) 50  34  11
```

"That's right" said the cop, "except you forgot the commas between the statement numbers."

Go To 48

Write a program for the following flow chart:



Go To 49

Reprinted from the abstract of SECOND ANNUAL CONFERENCE ON PROGRAMMING ETHICS, Washington, D. C., 1961.

The time has come for programmers to:

STOP adopting programs, written by other people, for unrelated problems.

STOP using facts when one can use scientific truths.

STOP demanding computer operations with extra sensory perception abilities.

STOP blaming the <u>computer</u> for writing <u>bugs</u> into the program.

STOP trying to make <u>complicated</u> problems out of <u>simple</u> ones.

In other words ----- STOP PROGRAMMING!

The Fortran program for the preceding example should resemble the following:

```
      PROGRAM TEST
C     TEST RUN NUMBER 100
      DIMENSION A(4, 5), DELTA(4, 5), BETA(4)
      I = 1
      J = 1
   10 IF (A(I, J)) 11, 12, 11
   11 DELTA(I, J) = BETA(I)/A(I, J)
      GO TO 20
   12 DELTA(I, J) = 0.
   20 IF (DELTA(I, J) - 6.2) 10, 30, 50
```

If your program differed significantly, review 43 through 47, and do 48 again.

Go To 50

Most Fortran programs involve looping, the repetition of groups of statements with varying parameters. The control statement involved is the "DO" statement. We begin with a very <u>simple</u> example.

```
    K = 0
    DO 303  J = 1, 3
    K = K + 2
303 SUM(J) = J ** 2 + K ** 2
```

The above DO statement calls for 3 loops, since the loop control sequencer "J= 1,3" indicates an initial value"1", and a terminal value,"3." The statements involved in the loop are <u>all</u> the statements following the DO down to and including the statement number 303. Thus:

First loop	$K = 0 + 2$ $SUM_1 = 1^2 + 2^2$
Second loop	$K = 2 + 2$ $SUM_2 = 2^2 + 4^2$
Third loop	$K = 4 + 2$ $SUM_3 = 3^2 + 6^2$

Go To 51

How about another example?

```
     DO 7 NN = 1, 4
     K(NN) = 3 ** NN + 5
     L(NN) = NN ** 4 + 5
7    M(NN) = K(NN) + L(NN)
```

The loop control sequencer "NN = 1, 4" indicates 4 loops since the initial and terminal values are 1 and 4 respectively. The "7" indicates that <u>all</u> statements following the DO statement, down to and including statement 7 are indicated in the loop.

Loop 1

$$K_1 = 3^1 + 5$$
$$L_1 = 1^4 + 5$$
$$M_1 = K_1 + L_1 = 14$$

Loop 2

$$K_2 = 3^2 + 5$$
$$L_2 = 2^4 + 5$$
$$M_2 = K_2 + L_2 = 35$$

Loop 3

$$K_3 = 3^3 + 5$$
$$L_3 = 3^4 + 5$$
$$M_3 = K_3 + L_3 = 118$$

Loop 4

$$K_4 = 3^4 + 5$$
$$L_4 = 4^4 + 5$$
$$M_4 = K_4 + L_4 = 347$$

Go To 52

A. Write the DO statement which will include the following statements in a loop using an initial value I = 1 and terminal value I = 200.

```
      BETA(I) = ALPHA(I) - D ** 2 + 7.0
      IF(BETA (I)) 10, 11, 16
   16 ALPHA(I) = F(I) - 4.2
```

B. Given the DO statement below, answer the following questions.

DO 100 JK = 5, 27

1. The "100" indicates that the loop should be executed 100 times (TRUE, FALSE)
2. The control sequencer, JK = 5, 27, indicates which one of the following:
   (1) JK takes on values 5, 6, 7, ---- 27.
   (2) J takes on values 1, 2, ---- 5 and K takes values 1, 2, --- 27.
   (3) J becomes 5 and K becomes 27.

Go To 53

For the first example A, of the preceding card, you should have the statement--

DO 16 I = 1,200

For the second example B, the correct answers are:

1. FALSE

2. JK takes values 5, 6, 7 --- 27

If you had some trouble with these two examples, GO TO 50. Otherwise,

Go To 54

Lest you become over-confident, let us warn you that there is more than meets the eye in the DO statement--that is more than has met <u>your</u> eye to this point!

The first of these "eye hiders" is the fact that the control sequencer may contain a third <u>variable</u> or <u>constant</u>, following the initial and terminal values. For example,

DO 17 I = 1, 51, 5

Here, the control sequencer, I = 1, 51, 5, indicates the following:

1 is to be the <u>INITIAL</u> value of I
51 is to be the <u>TERMINAL</u> value of I
5 is to be the <u>INCREMENTAL</u> value of I

When the INCREMENT is equal to 1 (the usual case) this third <u>variable</u> or <u>constant</u> <u>need not be</u> expressed. Thus I takes values 1, 6, 11, ------ 51, since the incremental value in the above example was 5.

Go To 55

The <u>index</u> of the "control sequencer" <u>must be a fixed point variable</u>. That is, the index must start with one of the letters I, J, K, L, M, N. (Actually, the compiled program works more efficiently if the <u>single</u> letters I, J, K, L, M, N are used for the index).

Of utmost importance is the fact that the two (or three) parameters (the initial value, the terminal value, and the increment) following the index of the control sequencer can be FIXED POINT VARIABLES as well as FIXED POINT CONSTANTS but <u>not</u> EXPRESSIONS. Side by side examples are:

```
DO 22  I = 1, 10              DO 22  I = N, 10
DO 56  J = 4, 100, 2          DO 56  J = N, K, 2
DO 30  L = 5, 89, 6           DO 30  L = N, K, J
```

Go To 56

There is still more to learn about the DO statement! For example,

The loop itself involves learning a few concepts.

One of these has to do with TERMINAL values of the index. The index is increased by the INCREMENT value at the bottom of the loop--after the last statement of the loop. If the index variable becomes greater than the terminal value after being incremented, the DO loop is terminated, and the index value is set to zero. The program then goes to the next sequential statement following the last statement of the DO loop. If a DO loop is left before the loop is terminated, the last value of the incremented index is retained.

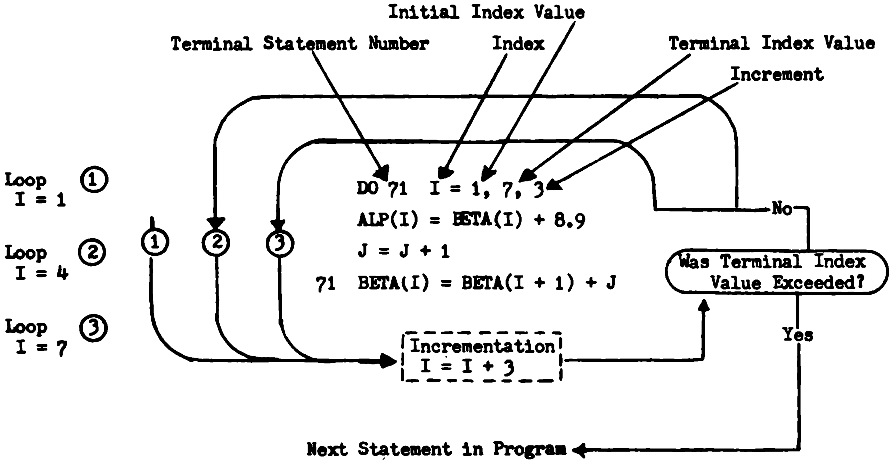
For example:

```
DO 17  J = 3, 7, 2
```

LOOPING continues for three loops (J = 3, J = 5, J = 7) and on normal exit J is set back to zero. However, if a inner exit (programmed) had occurred during the second loop, the last value, 5, would have been retained by J.

Go To 57

Diagramatically, we might show the looping concept as follows:



Go To 58

There are two very important restrictions concerning the last statement of a DO loop. First, there must be a statement number (in columns 1 - 5) for this statement. Second, the last statement can not be a control transfer type statement ("GO TO" or "IF").

However, there are instances where you will want to put an IF or GO TO statement at the end of your DO loop. If so, a special "do nothing" type statement, CONTINUE, is required to end the loop. Since this is the last statement of the loop, it must have the same statement number previously expressed in the DO statement. An example follows:

```
      DO 6 J = 5, 100
      B(J) = A(J) + TIME ** 3
      IF (B(J)) 10, 6, 6
    6 CONTINUE
```

Go To 59

Finally, DO loops can be contained within DO loops as long as the inner DO Loops are nested loops (contained within the outer loop). It is possible under this condition for several DO loops to terminate with the same statement. Also note that nested DO statements can not use identical index variables. The following diagrams indicate correct inner loops.

```
      DO 3 I = 1, 100

      DO 4 K = 1, 100
    4
      DO 5 J = 1, 100
    5
    3
```

```
      DO 3 I = 1, 100

      DO 3 K = 1, 100

      DO 3 J = 1, 100
    3
```

Go To 60

Now that all the rules are in, did we over DO it? The best way to find out is by examples such as the following.

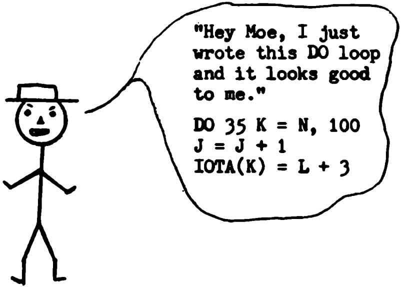
A programmer recently wrote the following portion of a problem.

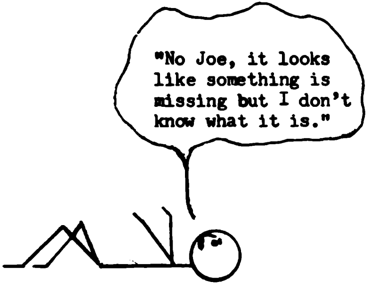
```
    PROGRAM TEMP
    DIMENSION W(500), Z(100), F(100)
    DO 15 I = 1, 100
    F(I) = DATA + 7.0
 12 G = G + 1.
    W(I) = F(I) + HL3 ** 2 - (Z(I) - G)
    IF (W(I)) 12, 12, 15
 15 CONTINUE
```

Did this programmer violate a DO loop principle? If you think he did, Go To 62. If you think the program is correct, Go To 61.

The statements were OK. It is good that you noticed that a CONTINUE statement must follow a transfer control statement when the transfer ends a DO loop.

The following incident took place one day:





If you agree with Joe, go to 63

If you agree with Moe, go to 64

YOU
JUST TOOK
A
DETOUR

Since the program
was correct
as
written

Notice that it was desired to have a control transfer (IF statement) at the end of the DO loop: Thus it was necessary to include CONTINUE as the last statement of the loop.

Go To 61

Roses are red,
Violets are blue;
Joe was WRONG
and so were YOU.

The last statement of a DO loop without a statement number is like a dog without a tail---something is missing!

Go To 64

Moe was right--and so were you.

THE STATEMENT NUMBER OF THE LAST STATEMENT OF A DO LOOP CAN NOT BE MISSING!

There are two errors in the following example. Find them.

```
    DO 32 I = J, J + 5
32  N(I) = M(I) - I
    DO 14 K = 1, 10
    DO 16 L = 1, 10
14  GAMMA(K) = D(K) + F
16  D(L) = F - 5.2
```

Go To 65

The two errors in the preceding example were:

1. The TERMINAL value parameter of a DO loop cannot be an expression (J + 5)--it is limited to FIXED POINT VARIABLES or FIXED POINT CONSTANTS.

2. The inner DO loop (DO 16 L = 1, 10) is <u>not</u> nested since statement 16 is not included within the outer DO loop (DO 14 K = 1, 10).

If you missed (1) above, REVIEW 55, then
If you missed (2) above, REVIEW 59, then
If you had then both correct (teacher's pet)
Go To 66

Some goofs are self evident--yet easy to make. Following are examples:

```
DO  12  A = 1, 10
```

DO loop index must be a FIXED POINT VARIABLE

```
Outer   DO 40 I = 1, 9, 2
Inner   DO 20 I = 1, 10
   20   BETA(I) = GAMMA(I) + 7.0
   40   ALPHA(I) = BETA(I) + 50.
```

Index of inner DO loop can <u>not</u> be same as index of outer loop.

Go To 67

Three control statements remain. These are "duck soup" for "pros" who have progressed this far!

First, there is the PAUSE statement, Example: PAUSE 712
This will cause the program to PAUSE, indicate the PAUSE number, and permits the operator to make one of the following decisions: (1) abandon the program, (2) proceed with no manual changes. The number following the PAUSE (up to 5 digits are permitted) enables the PAUSE statement to be uniquely identified. By using PAUSE statements, one can periodically determine where he is in the program.

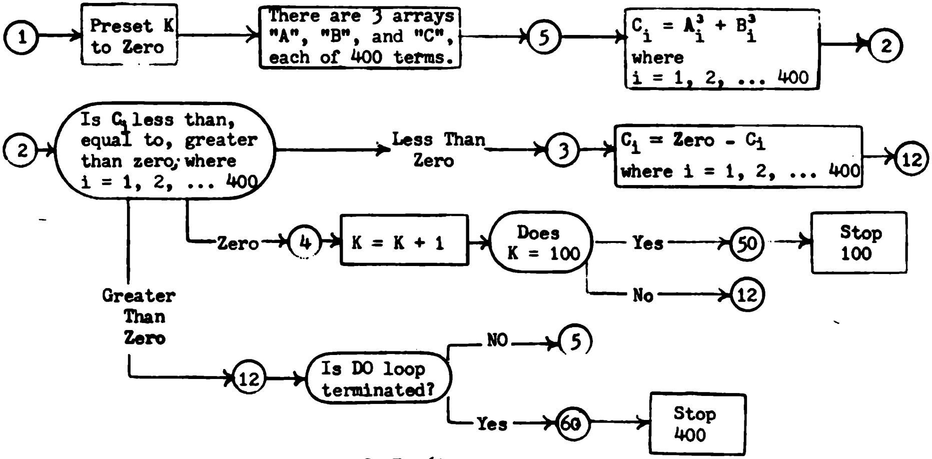
Second, there is the STOP statement. Example: STOP 355
This causes the program to be abandoned after indicating the STOP number. The number (up to 5 digits) provides identification in case one has a STOP at various places in the program.

Third, there is a REWIND statement. Example: REWIND 3
This will cause magnetic tape number 3 to be rewound to its starting position.

Go To 68

The next few cards will attempt to lead you through a General Review. Good luck! Write statements for the following Flow Chart.

Flow Chart of "TEMPEST" Program



Go To 69

How does your version stack up with the following?

```
      PROGRAM TEMPEST
      DIMENSION A(400), B(400), C(400)
      K = 0
      DO 12 I = 1, 400
      C(I) = A(I) ** 3 + B(I) ** 3
      IF (C(I)) 3, 4, 12
    3 C(I) = 0. - C(I)
      GO TO 12
    4 K = K + 1
      IF(K - 100) 12, 50, 12
   12 CONTINUE
      STOP 400
   50 STOP 100
```

You are permitted some slight variations from the above. For example, K = 0 can appear before the DIMENSION statement. Also, Statement 50 can be placed <u>inside</u> the DO loop. After reviewing your errors,

Go To 70

The following frequency table, containing FIXED POINT CONSTANTS, lists 100 corresponding values of K and L.

TABLE

K_i	L_i
K_1	L_1
K_2	L_2
K_3	L_3
⋮	⋮
K_{100}	L_{100}

MEAN and VARIANCE are found using the following rules; where all summations range from i = 1, to i = 100.

$$\text{MEAN} = \frac{\sum K_i L_i}{N}$$

$$\text{VARIANCE} = \frac{\left(\sum L_i K_i^2 - \frac{(\sum K_i L_i)^2}{N}\right)}{N - 1}$$

where $N = \sum L_i$

Write the program, "MEASURE", to find MEAN and VARIANCE in FIXED POINT.

Go To 71

Program "MEASURE" would appear similar to the following:

```
      PROGRAM MEASURE
      DIMENSION K(100), L(100)
      N = 0
      MEAN = 0
      JVARNCE = 0
      DO 5 I = 1, 100
      N = L(I) + N
      MEAN = K(I) * L(I) + MEAN
    5 JVARNCE = L(I) * K(I) ** 2 + JVARNCE
      MEAN2 = MEAN ** 2
      MEAN = MEAN/N
      JVARNCE = (JVARNCE - MEAN2/N)/(N - 1)
```

Note MEAN2 is assigned to $\left(\sum_{i=1}^{100} K_i L_i\right)^2$; fixed point VARIANCE = JVARNCE

Study the above! You may have had another correct version. If so, and if it is more efficient, please send a copy to Senior Senator Humphrey, St. Paul 1, Minnesota.

Go To 72

A sine function is evaluated by using a truncated Chebyshev Polynomial of the following form:

$$\text{SINE } X = C_1 X + C_3X^3 + C_5X^5 + C_7X^7 + C_9X^9 + C_{11}X^{11}$$

where X is the argument in radians, and C_1, C_3, --- C_{11} are Floating Point Constants.

Assuming 200 arguments located in array, X, and the constants C_1, C_3 --- C_{11} located in the 6 member array C, write a Fortran program to compute 200 SINE values.

Go To 73

Two solutions are given. The first uses a nested form of the polynomial.

```
      PROGRAM SINES
C     USING CHEBYSHEV SERIES NESTED
      DIMENSION X(200), C(6), SINE(200)
      DO 10 I = 1, 200
      S = X(I) ** 2
   10 SINE(I) = X(I)*(C(1)+S*(C(2)+S*(C(3)+S*(C(4)+S*(C(5)+S*C(6))))))
```

Using the polynomial in its original form provides another solution.

```
      PROGRAM SINES
C     USING CHEBYSHEV SERIES NOT NESTED
      DIMENSION X(200), C(6), SINE(200)
      DO 10 I = 1, 200
      SINE(I) = 0.
      L = 0
      DO 10 K = 1, 11, 2
      L = L + 1
   10 SINE(I) = SINE(I) + C(L) * X(I) ** K
```

As you see, there's more than one way to "skin the sine". Study the above and then

Go To 74

It was a dismal day--even the local pub, down the street, looked deserted. I paced nervously back and forth--there was so little time to save Detson!

"My dear Watson", said Holmes, looking up from the clues of the Detson case, "quit your pacing and read this note taken from the body of Lippy Fug."

Dear Holmes,

Reading this means you finally got me! But you're a gentleman, Holmes, and I always give a gentleman a break. Calling a certain number can save Detson--and earn you his fee. So I leave a puzzler, me lad. The sum of all the possible products, taken two at a time, of all the convict numbers housed at Ole Bailey and Ole Gimey is first needed. This sum divided by 12, rings the right bell, old chappie.

"You see, my dear Watson", continued Holmes, "I called the warden and found Ole Bailey has 100 convicts numbered 50 to 149; Ole Gimey contains 100 convicts numbered 1 to 100".

"But Holmes, I shouted, you know and I know that 100 by 100 gives 10,000 possible products which must be summed and then divided by 12. Detson dies in two hours, we can't possibly do it by then.

"Calm yourself, dear Watson", said Holmes. "I just called the firm of Control Data, Ltd. They have a gadget called the 1604 which can get us the answer. However the problem must be written in Fortran language first. They told me a student <u>now</u> learning Fortran should be able to write this program. I am waiting for his call now, and as soon as he finishes, we will all

Go To 75

```
      PROGRAM DETSON
C     CALL NUMBER LOCATED AT RING
      DIMENSION BAILEY(100), GIMEY(100)
      RING = 0.
C     GENERATE OLE BAILEY CONVICT NUMBERS
      BAILEY(1) = 50.
      DO 20 I = 2, 100
   20 BAILEY(I) = BAILEY(I - 1) + 1.
C     GENERATE GIMEY CONVICT NUMBERS
      GIMEY(1) = 1.
      DO 30  I = 2, 100
   30 GIMEY(I) = GIMEY(I - 1) + 1.
      DO 40 I = 1, 100
      DO 40 J = 1, 100
   40 RING = RING + BAILEY(I) * GIMEY(J)
      RING = RING/12.
      STOP
```

(Note: We have inserted two comments to indicate what the program is doing at certain times)

Go To 76

The Detson case ended not only Lippy Fug's secret, it also ended the review of the first phase of this course (definitions, symbols, and control statements).

Incidentally if you objected to the Detson case as being unrealistic because telephones were not available in Sherlock Holmes' day, let us point out that 1604's by Control Data Corporation were also unavailable.

The point we want to make is: If you had trouble with <u>any</u> of the review exercises up to this point---REVIEW! REVIEW! REVIEW! Then try the exercises again. If you then feel like Floyd Patterson, after his <u>first</u> fight with "Ingy", REVIEW some more.

If you feel like Floyd after his <u>second</u> meeting with "Ingy,"

Go To 77

As the art of programming developed, it became evident that certain mathematical functions (for example; sine, cosine, logarithm, square root, etc.) were used several times in the same program or they were used in many programs. Programmers soon realized that much time and inconvenience could be saved if these functions were written as packaged routines which could be referenced by main programs when desired. These packaged routines became known as SUB-ROUTINES.

As the number of these SUB-ROUTINES increased, it was a common procedure to store them on some on-line peripheral input device and to bring them into the computer when they were referenced by a program. Magnetic tape units were particularly effective in this respect.

The 1604 Fortran System uses such a tape and it is known as the REFERENCE LIBRARY. Tape Unit 1 is reserved for the REFERENCE LIBRARY. This tape contains mathematical SUB-ROUTINES which are always available for the programmer to use.

Go To 78

The following SUB-ROUTINES are available from the REFERENCE LIBRARY.

ACOSF	arc cosine	ABSF	absolute value
ASINF	arc sine	EXPF	"e" to a power
ATANF	arc tangent	SINH	hyperbolic sine
SINF	sine	COSH	hyperbolic cosine
COSF	cosine	TANH	hyperbolic tangent
TANF	tangent	LOGF	natural log
SQRTF	sq. root	LOG10F	common log

These are referenced as shown in the following statement examples:

Example 1

```
Y = SINF(X) + 7.7
```

Name of Sub-routine (must be one of the forms given above)

Argument must be in parentheses; and must be either a FLOATING POINT: VARIABLE, CONSTANT, or EXPRESSION.

Example 2

```
ALPHA = BETA(I) + LOGF(X + Y)
```

Go To 79

The following examples and their solutions should help.

Given: $Ax^2 + Bx + C = 0$. Solve for positive x.

```
X = (-B + SQRTF (B**2. - 4. *A*C))/(2.*A)
```

Given: Secant B + sin α = Y. Write in Fortran.

```
Y = 1./COSF(BETA) + SINF(ALPHA)
```

Given: $G = \log_e (X + \sqrt{X^2 - 1})$. Write in Fortran.

```
G = LOGF(X + SQRTF(X**2 - 1.))
```

Go To 80

Subroutines are often needed which are not in the REFERENCE LIBRARY, or you may want to sub-divide your program into sub-programs. In either case, you will need to know how to write subroutines, where to locate them in the overall program, and how to call them up when needed.

Let's first learn how to write a SUBROUTINE! One simply writes an identification statement (see 'a' below) which names the SUBROUTINE and lists its arguments. Following this are the conventional statements (which you have learned) and these make up the body of the SUBROUTINE (lines below). Within the SUBROUTINE, you may desire a RETURN statement (see b). A RETURN statement will provide an inner exit from the SUBROUTINE-- before reaching the normal (regular) exit. The normal exit is formed by writing one END statement (see c), as the last statement of the SUBROUTINE. Study the example.

```
a.        SUBROUTINE BETSY (ALPHA, NET)
          _____________________________
          _____________________________
          _____________________________

b.        RETURN
          _____________________________
          _____________________________
c.        END
```

Go To 81

Having written a SUBROUTINE, where does one place it in the overall program? The correct order of placement is: PROGRAM NAME, main program containing conventional statements, the main program END statement, first SUBROUTINE with its END statement, second SUBROUTINE with its END statement, the next SUBROUTINE with its END statement, etc. Finally, an extra END statement, as a final statement, indicates the conclusion of the overall program.

Every calendar lists a correct order of months and days. Thus it would hardly be "kosher" to try to place June after October. Fortran is like this in that the order above must be followed.

Lest you forget, we re-emphasize this order by having you

Go To 82

The diagram indicates the organization of a complete example program.

PROGRAM EXAMPLE → Program NAME Statement

Main Program
Statements

END
SUBROUTINE ONE(X,Y)

Subroutine
Statements

END
SUBROUTINE TWO(G, ALPHA)

Subroutine
Statements

END
END → LAST Statement of Program

END Statements

Sub-Routine NAME Statements

Go To 83

One phase of SUBROUTINE usage remains -- how to reference (CALL) the SUBROUTINE from the main program! This is done by a CALL statement which specifies the *name* and *arguments* of the SUBROUTINE. After the subroutine is executed, control goes to the statement that follows the CALL statement.

Of particular importance are two requirements: first, the *number* of arguments in the CALL statement and the identification statement of the SUBROUTINE must be equal; second, the *modes* (FIXED or FLOATING) of the arguments in the CALL statement and the corresponding arguments of the SUBROUTINE identification statement must be the same. For example:

```
        CALL GENER (ALPHA, BETA, JET)

           SUBROUTINE GENER (X,   Y,   N)
```

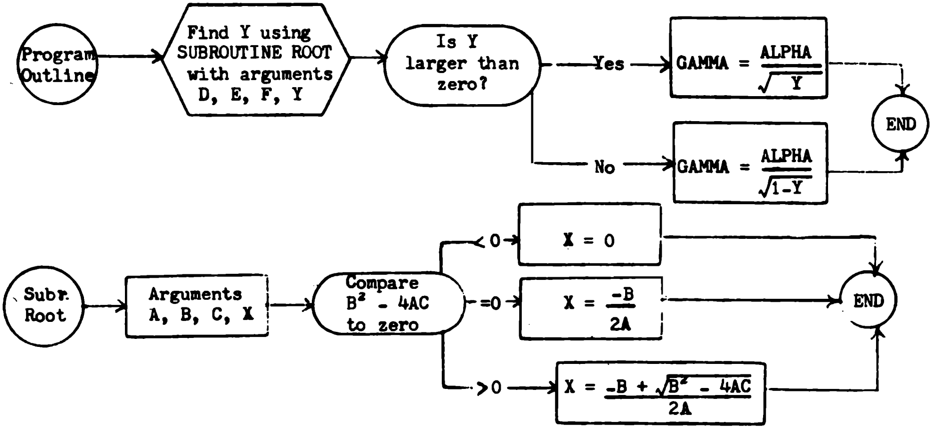
Note:

Each contains 3 arguments and corresponding arguments agree in mode.

The SUBROUTINE arguments (X, Y, N in this example) are DUMMY VARIABLES--they always take on values of the corresponding arguments in the CALL statement.

Go To 84

Can you write a program involving a SUBROUTINE? Use the flow chart below:



Go To 85

```
      PROGRAM OUTLINE
      CALL ROOT (D, E, F, Y)
      IF (Y) 10, 10, 20
   10 GAMMA = ALPHA/SQRTF(1. - Y)
      GO TO 30
   20 GAMMA = ALPHA/SQRTF(Y)
   30 END
      SUBROUTINE ROOT (A, B, C, X)
      IF (B ** 2 - 4. * A * C) 1, 2, 3
    1 X = 0.
      RETURN
    2 X = (-B)/(2. * A)
      RETURN
    3 X = ((-B) + SQRTF (B ** 2 - 4. * A * C)) / (2. * A)
      END
      END
```

How did you do? If you had trouble, try reviewing 80-84.

Go To 86

Two arrays A and B each contain 100 terms. A subroutine, TEST, examines <u>each</u> set, made up of two corresponding terms (one from each array), and determines which is the larger. If the term from A is larger than the corresponding term from B, the subroutine assigns the digit 1 to J. If the term from A is smaller, the subroutine assigns the digit 0 to J. If the terms are equal, B is assumed the larger.

The main program, TESTER, always computes the following:

X = sine of the larger term of a set

Y = the square root of the smaller term of a set

Z = the square root of the larger + the cube of the smaller.

Write the program, TESTER, including the subroutine, TEST, which will determine X, Y and Z for each of the 100 sets.

Go To 87

```
      PROGRAM TESTER
      DIMENSION A(100), B(100), X(100), Y(100), Z(100)
      DO 15 I = 1, 100
      CALL TEST (A(I), B(I), K)
      IF (K) 14, 14, 12
   12 X(I) = SINF (A(I))
      Y(I) = SQRTF (B(I))
      Z(I) = A(I) ** 2 * B(I) ** 3
      GO TO 15
   14 X(I) = SINF (B(I))
      Y(I) = SQRTF (A(I))
      Z(I) = B(I) ** 2 + A(I) ** 3
   15 CONTINUE
      END
      SUBROUTINE TEST (A, B, J)
      IF (A - B) 5, 5, 6
    5 J = 0
      RETURN
    6 J = 1
      END
      END
```

If you found significant mistakes, review 80 - 84; otherwise,

Go To 88

How do you feel after the last example? If you don't feel so good, remember-----

Rome was not built in a day,
Rome was not torn down in a day,
But Rome was conquered!

Once more we emphasize the relationship between arguments in a CALL statement and corresponding arguments in a SUBROUTINE name statement. For example!

CALL JOSEY (L(I), X + Y, 2.6) and SUBROUTINE JOSEY (M, A, X)

(1) There are 3 arguments in each--the NUMBER of arguments must be equivalent.

(2) L(I) and M are FIXED POINT arguments, X + Y and A are FLOATING POINT, and 2.6 and X are FLOATING--CORRESPONDING arguments must be of the same mode.

(3) L(I) is a FIXED POINT variable, X + Y is a FLOATING EXPRESSION, and 2.6 is a FLOATING CONSTANT -- the CALL statement arguments can be FIXED or FLOATING VARIABLES, CONSTANTS, or EXPRESSIONS.

(4) THE SUBROUTINE arguments are DUMMY VARIABLES. As such they can only be FIXED or FLOATING <u>non</u>-subscripted VARIABLES. These DUMMY VARIABLES take on the values of the CORRESPONDING arguments in the CALL statement.

Go To 89

The following three examples summarize the relationship between arguments of CALL statements and SUBROUTINE name statements.

CALL MATE (X, 5, ZEB) and SUBROUTINE MATE (A,N,C)

(1) Same number of arguments (3 each).

(2) Corresponding arguments are of the same mode (X and A, FLOATING; 5 and N, FIXED; ZEB and C, FLOATING)

CALL BENTLY (LOT (I), A + B) and SUBROUTINE BENTLY (JOT, VETO)

(1) Same number of arguments (2 each).

(2) Corresponding arguments are of the same mode - LOT(I) and JOT are FIXED; A + B and VETO are FLOATING.

CALL FIX and SUBROUTINE FIX

(1) Number of arguments--NONE. This SUBROUTINE needs no arguments since it is constructed to be self contained. It may perform an action completely unconnected to the main program. If it must communicate in some way with the main program, this can be done through the use of COMMON storage which is explained later.

Go To 90

If all the "Smiths" in a city lived in one house, two forms of "sharing" would be present. First all Smiths share the same house or location; second, all share the same name.

Efficient programming is possible if the same two forms of sharing exist between sub-divisions of a program. For example, if programs, JOHN and JANE each use the same variables--why not share these variables? Or if JOHN and JANE each use temporary locations in arriving at final results--why not share these temporary locations?

The Fortran COMMON statement provides for sharing variables and locations, as;

Example 1

Main program, WALK, and subroutine, RUN use the same variables A and B.

```
PROGRAM WALK
COMMON A, B
.   .   .
.   .   .
.   .   .
SUBROUTINE RUN (X, Y)
COMMON A, B
.   .   .
.   .   .
.   .   .
```

Example 2

Main program, AWAKE, and subroutine, SLEEP use the same temporary array, DATA.

```
PROGRAM WALK
DIMENSION DATA (1000)
COMMON DATA
.   .   .
.   .   .
.   .   .
SUBROUTINE SLEEP (X, Y, D)
DIMENSION DATA(1000)
COMMON DATA
.   .   .
.   .   .
```

Go To 91

When COMMON is used in sub-divisions of the same program, assignments in COMMON storage are made from the same reference point. Thus in the example below, COMMON B, in subroutine GENER, assigns B to the <u>same</u> locations as were assigned Y by COMMON Y of main program. One can see that relative order of variables and of array names is important. Also note that array names are listed as <u>non-subscripted</u> variables.

```
PROGRAM PIECE
DIMENSION X(200), Y(200)
COMMON Y
CALL GENER (X)
.     .     .     .
.     .     .     .
.     .     .     .
SUBROUTINE GENER (A)
DIMENSION A(200), B(200)
COMMON B
DO 15 I = 1, 200
A(I) = B(I) + A(I)

.     .     .     .
.     .     .     .
.     .     .     .
.     .     .     .
```

In this example, both the main program, PIECE, and the subroutine, GENER use the same 200 Y values. Thus 200 storage locations are shared through COMMON rather than the main program and subroutine each requiring 200 locations. (In the main program these variables are called Y and in the subroutine they are called B)

Go To 92

"ELEGANT" use of COMMON can lead to interesting results. For example:

```
PROGRAM HUMP
COMMON X, Y
  .   .   .
  .   .   .
CALL NOARG
  .   .   .
  .   .   .
  .   .   .
END
SUBROUTINE NOARG
COMMON A, B
  .   .   .
  .   .   .
END
END
```

Subroutine variables A and B, are assigned same locations held by variables X and Y of the main program. Thus one is able to transfer results from a subroutine to the main program <u>without</u> having to list arguments of the subroutine.

```
PROGRAM MIX
COMMON X, Y
  .   .   .
  .   .   .
CALL MIXER
  .   .   .
  .   .   .
  .   .   .
END
SUBROUTINE MIXER
COMMON Y, X
  .   .   .
  .   .   .
END
END
```

Subroutine variables Y and X, of the subroutine are assigned to respective locations previously held by X and Y of the main program. The result is to interchange names of variables from main program to subroutine.

Go To 93

In summary, the COMMON statement is given by simply writing "COMMON" followed by a list of FIXED or FLOATING VARIABLES. If the variable name represents an array, the array name must appear without a subscript. A DIMENSION statement given previously provides for the array range. Thus if the array A(200,200) is to be assigned to COMMON, one would write:

```
DIMENSION A(200,200)
COMMON A
```

Keep in mind, each COMMON statement operates within its own individual program--that is, a sub-program, such as a SUBROUTINE, will require its own COMMON statement. Also, the order and number of assignments is crucial. Since programs and sub-programs are compiled independently, the use of more than one COMMON statement within the same program or same sub-program will make additional assignments in succession without overlapping previous assignments. However, COMMON statements appearing in each sub-program will make assignments which start from the same COMMON reference point. The degree of overlapping will depend upon the list order of the COMMON statements.

Go To 94

Try the following exercise!

There are 3000 values stored at array, TANG. Program, TANGENT, computes the tangent of each value and stores the function results, Y, at the same storage locations reserved for TANG. The following program was written using the tangent function subroutine, TANF, from the Reference Library.

```
      PROGRAM TANGENT
      DIMENSION TANG(3000), Y(3000)
      COMMON TANG, Y
      DO 20 I = 1, 3000
   20 Y(I) = TANF(TANG(I))
      END
      END
```

If you think the above program is correct, Go To 96.

If you think it is incorrect, Go To 95.

You deserve a Gold Medal for excellance! The solution is incorrect, and even the irrevelent Y (for results), given in the problem statement, did not cause you trouble. The correct solution would not require the use of Y or COMMON, and could be written as:

```
      PROGRAM TANGENT
      DIMENSION TANG(3000)
      DO 20 I = 1, 3000
 20   TANG(I) = TANF(TANG(I))
      END
      END
```

Try writing program, MEASURE, described below.

There are 1000 positive two digit numbers at array, DEBIT. A subroutine, MEAN, computes the average, AVER, of all the numbers in the array that are less than 50.

The main program perserves DEBIT, and forms a new array, BALANCE, by adding the average, AVER, (found in the subroutine) to the first 500 numbers of DEBIT and subtracting AVER from the last 500 numbers of DEBIT.

Go To 97

The stated problem wanted each initial value at TANG to be replaced by a new value. In Fortran, a procedure such as this is easily accomplished by writing the variable to be replaced on the left side of the equal (=) sign and the modified form of the variable on the right side. For example, to replace each Z by Z + A, one simply writes:

Z = Z + A

Therefore, in the previous example, you should have recognized the irrevelency of Y--knowing that to replace TANG, all that was needed was a statement similar to:

TANG(I) = TANF(TANG(I))

The second misconception may have come from a misunderstanding of COMMON. The statement "COMMON TANG, Y" does not assign arrays TANG and Y to the same location. Rather, it assigns EACH to locations--in the sequential order: array TANG, array Y. These arrays may then be referenced by other COMMON statements that appear in sub-programs (subroutines) of the total program. In the given example, there were no sub-programs, and COMMON had no useful purpose for being there.

Now, try re-writing the previous example, correctly and then,

Go To 95

```
      PROGRAM MEASURE
      DIMENSION DEBIT(1000), BALANCE(1000)
      COMMON DEBIT
      CALL MEAN(AVER)
      DO 5 I = 1,500
    5 BALANCE(I) = DEBIT(I) + AVER
      DO 10 I = 501, 1000
   10 BALANCE(I) = DEBIT(I) - AVER
      END
      SUBROUTINE MEAN (AVER)
      DIMENSION DEBIT(1000)
      COMMON DEBIT
      CTR = 0.
      AVER = 0.
      DO 15 I = 1, 1000
      IF (DEBIT(I) - 50.) 6, 15, 15
    6 CTR = CTR + 1.
      AVER = AVER + DEBIT (I)
   15 CONTINUE
      AVER = AVER/CTR
      END
      END
```

Note: By assigning the array, DEBIT, to COMMON, 1000 locations are saved since both the main program and the subroutine are able to use the same locations.

Go To 98

How about another example?

Assume there are 1000 positive two digit numbers (all integers) at array, LIST. A subroutine DIGIT, consecutively adds 10 to each LIST term until the accumulated sum LSUM, for that term is greater than 100. The number of additions of 10 for that term are also kept in counter, KTR.

The main program uses LSUM and KTR of each term (found in the subroutine) to compute the following for each term:

(1) The right most digit (units' digit) of the LIST term.

(2) The quotient, J, found by dividing the right most digit(found in step 1) by KTR.

(3) The value, Y, formed by taking the sine of the quotient found in step 2.

Try writing this program in two ways: first, by making use of COMMON; second, by not using COMMON.

Go To 99

(using COMMON)

```
      PROGRAM FACTOR
      DIMENSION LIST (1000), Y(1000)
      COMMON LIST, LSUM, KTR
      DO 50  I = 1, 1000
      CALL DIGIT(I)
      FJ = (LSUM - 100)/KTR
50    Y(I) = SINF(FJ)
      END
      SUBROUTINE DIGIT(J)
      DIMENSION LIST (1000)
      COMMON LIST, LSUM, KTR
      LSUM = LIST(J)
      KTR = 0
 2    LSUM = LSUM + 10
      KTR = KTR + 1
      IF(LSUM - 100) 2, 2, 4
 4    END
      END
```

Two points are worthy of note!

Variable, J, is made, FJ, since the NEXT STEP involves the SINE of FJ and the sine routine requires a FLOATING POINT argument.

Transfer of data between main program and subroutine is made by putting LIST, LSUM, and KTR in COMMON and by setting up a needed link between I and J in the CALL and SUBROUTINE statements.

Go To 100

(without using COMMON)

```
      PROGRAM FACTOR
      DIMENSION LIST(1000), Y(1000)
      DO 50 I = 1,1000
      CALL DIGIT (LIST(I), LSUM, KTR)
      FJ = (LSUM - 100)/KTR
   50 Y(I) = SINF(FJ)
      END
      SUBROUTINE DIGIT (J, LSUM, KTR)
      LSUM = J
    2 LSUM = LSUM + 10
      KTR = KTR + 1
      IF (LSUM - 100) 2, 2, 4
    4 END
      END
```

In this method, the communication links between main program and subroutine are established by the corresponding variables listed in the CALL and SUBROUTINE statements. Thus LIST(I) goes to J,LSUM to LSUM, and KTR to KTR.

In this example only 2 more locations are required by not using COMMON. Recalling a previous example, where use of COMMON saved 1000 locations emphasizes that COMMON is more useful in some problems than others.

Go To 101

Handy Rules of Thumb:

First draw the curves -- then plot the data.

In debugging any type of program, no corrections can be made correctly after 4 p.m. Friday.

The correct corrections will be self-evident at 9:01 Monday morning.

"Bugs" which would normally take one day to find will take five when you are in a hurry.

Experience is directly proportional to computer time wasted.

What's in a name?
A rose by any other name
Would smell as sweet!

Such said the poet long ago. How about the programmer using Fortran? An identification statement is: SUBROUTINE JESSIE (NO, NEVER, MAYBE). Can we call "JESSIE" as follows.

```
CALL JESSIE (YES, PERHAPS, MAYBE)
```

If you think this is O. K.,
Go To 102

Otherwise
Go To 103

<u>Men</u> Claim:

"When a lady says <u>NO</u>, she means yes. When a lady says <u>never</u>, she means perhaps," but in this case NO can <u>not</u> mean YES and Never can <u>not</u> mean Perhaps, since FIXED POINT and FLOATING POINT variables can not be mixed in the calling sequence of subroutines. Arguments of subroutines must agree in mode and number with the dummy variables of the subroutine identification statement.

Go To 103

Self-evident-appearing facts are often misleading. Thus a lady who puts her arm out the car window may not be indicating a left turn--perhaps it only means the window is down. The fact that you correctly answered the previous question doesn't mean you know all about SUBROUTINES--so let's try another!

Would you say there are one or two rules that appear to be broken in the following?

Part of Main Program	Part of Sub Program
DIMENSION A(100), B(100)	SUBROUTINE EVIDENT (ALPHA, BETA, GAMMA)
COMMON J, B, K + M	. . . .
CALL EVIDENT (X, Y, B, 3)	. . . .

If you think there was one rule broken, Go To 105
If you think there were two rules broken, Go To 104

Two rules were broken--the number of arguments must agree, and an expression (K + M) can not appear in a COMMON statement.

"Can I take your order?" asked the counter-man.

"Yes," I said, "make it a cheese sandwich and a piece of pie."

"Pie and cheese sandwich", shouted the man.

"No," I retorted, "it's cheese sandwich, then pie--I don't want pie first".

Order is important in Fortran too. For example, the list of variables in a CALL statement establishes an order of modes (FIXED or FLOATING) which must be maintained by the corresponding dummy variables listed in the subroutine identification statement.

The same type of order by modes should be maintained between variables of COMMON statements in a main program and its sub-programs COMMON statements.

CALL JOE (I, J, A)	COMMON M, J, E (main program)
SUBROUTINE JOE (N, K, D)	COMMON N, K, G (subroutine)

Thus, in each set above, the first two corresponding variables are FIXED POINT, the third is FLOATING.

Go To 106

Perhaps she was making a right turn! There were two rules broken.

Expressions are not permitted in COMMON statements--only variable names. Thus COMMON J, B, K + M is not allowed.

The other error occurred in the subroutine calling sequence. CALL supplies four arguments (X, Y, B, 3) while the subroutine EVIDENT is prepared to accept only three (ALPHA, BETA, GAMMA). Also don't forget that there must be an agreement of mode, thus statements similar to the following would correct this error.

```
CALL EVIDENT (X, Y, B, 3)
SUBROUTINE EVIDENT (ALPHA, BETA, GAMMA, N)
```

Go To 104

A main program uses the following statement:

COMMON A, B, I, D

A subprogram uses three variables, BETA, GAMMA, and X. BETA uses the data assigned to "A" in the main program. GAMMA uses the data assigned as "D" in the main program. In addition, the variable "X" in the subprogram is referenced through COMMON by the main program.

Assume the COMMON statement of the subprogram is written as either 1,2, or 3;

1. COMMON BETA, X, I, GAMMA
2. COMMON BETA, GAMMA, X
3. COMMON BETA, B, X, GAMMA

Choose the correct one above, and depending on your choice (J=1, 2, or 3);

Go To (107, 109, 108), J

THE LAW OF THE TOO SOLID GOOF

In any program, the part that is most obviously correct beyond all need of checking is the part that is wrong.

Corollary I   No one whom you ask will see it either

Corollary II   Everyone who stops by with unsought advice will see it immediately.

Shed a tear for Geary,
Hang the head for Ned,
COMMON goofs brought reproofs,
With censure gone, let's go on!

A SUBROUTINE, "TASK" is the <u>last</u> subprogram of a larger program. It uses three arrays, AID, APT, and ALERT, each of 400 terms kept in COMMON storage. Statement number 62 in the SUBROUTINE, examines the value of the expression formed by the sum of the last terms of arrays AID and APT. If zero, control goes to statement 72, if not zero, an exit is made from the SUBROUTINE. Statement 72 of the SUBROUTINE examines the value of another expression formed by the sum of the first terms of arrays AID and ALERT. If not zero, an exit from the SUBROUTINE is made. If zero, control goes to a STOP.

Write as many Fortran statements as possible from the above description. Then,

Go To 110

COMMON BETA, B, X, GAMMA

The statement above is not correct since the FLOATING POINT variable, X, is being assigned to a location defined as FIXED POINT, I, in the main program.

It is very important that you keep the <u>order</u> and <u>mode</u> consistent between the common statements of the main program and the common statement of the subprogram.

The correct COMMON statement for the example problem would be:

COMMON	A, B, I, D	(Main Program)
COMMON	BETA, X, I, GAMMA	(Subprogram)

Go To 107

COMMON BETA, GAMMA, X

The statement above is not correct since the variable GAMMA is assigned the same storage location as "B", instead of "D", in the main program. Also since "X" is FLOATING POINT, it can not be assigned to "I" which is FIXED POINT.

It is very important that you keep the order and mode consistent between common statements of the main program and common statements of the subprograms.

The correct COMMON statement for the example program would be.

```
COMMON  A, B, I, D                      (Main program)
COMMON  BETA, X, I, GAMMA               (Subprograms)
```

Go To 107

```
      SUBROUTINE TASK
      DIMENSION AID (400), APT (400), ALERT(400)
      COMMON AID, APT, ALERT
62    IF (AID(400) + APT(400)) 65, 72, 65
65    RETURN
72    IF (AID(1) + ALERT(1)) 65, 75, 65
75    STOP
      END
      END
```

Some variation of the above solution is possible. For example a second RETURN statement could have been used.

We call your attention again to the fact that a RETURN statement provides a subroutine inner exit preceding the normal exit designated by the subroutine END statement. Also notice that the second END statement is required to indicate to the compiler that this is the last statement of the total program.

Go To 111

COMPUTING CENTER HAZARDS

Necktie caught in magnetic tape handlers during rewind

Consumptive tendencies induced by air-conditioning

Sleep-walking maintenance men and programmers

Being run down by spectators

Up to this point, approximately three-fourths of the study course has been presented. Only INPUT-OUTPUT remains! The following exercises review information given up to this time.

The cards of a Fortran program (excluding Input-Output statements) fell on the floor. After picking them up, the following order was found. Can you unscramble the "pick-up order" to the correct order?

```
      IF (DATA) 5, 5, 3
      PROGRAM DROPSY
    5 END
    3 DO 10 I = 1, 100
      COMMON A, B, X
      DIMENSION A(100), B(100), X(100, 100)
      END
      DO 10 J = 1, 100
   10 X(I, J) = B(I) + A(J)
```

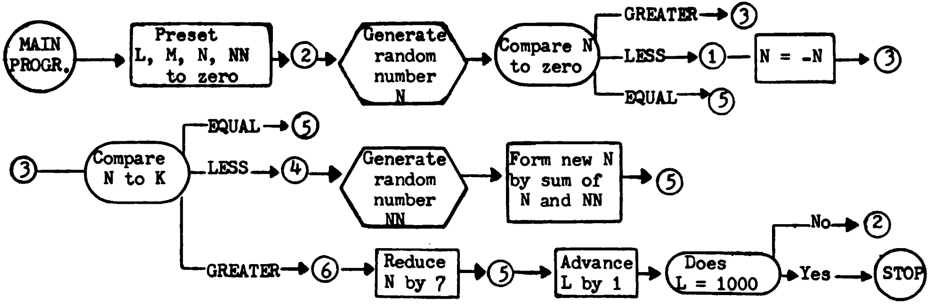
Go To 112

```
      PROGRAM DROPSY
      DIMENSION A(100), B(100), X(100, 100)
      COMMON A, B, X
      IF (DATA) 5, 5, 3
    3 DO 10 I = 1, 100
      DO 10 J = 1, 100
   10 X(I,J) = B(I) + A(J)
    5 END
      END
```

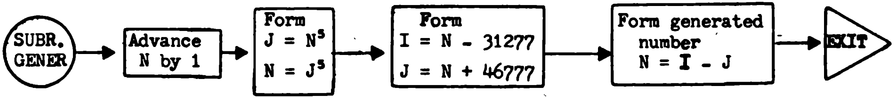
Are you wondering why there is a need for the COMMON statement above? If you are, so are we! However, it is left there as an example of superfluous statements, a "freeloader",--most programs usually have a few!

Go To 113

Using the flow chart as a guide, write the required Fortran statements.



The NUMBER GENERATOR subroutine indicated above is:



Go To 114

ROTATE 90 DEGREES

```
      PROGRAM EXAMPLE
      L = 0
      M = 0
      N = 0
      NN = 0
2     CALL GENER (N)
      IF (N) 1, 5, 3
1     N = -N
3     IF (N - K) 4, 5, 6
4     CALL GENER (NN)
      N = N + NN
      GO TO 5
6     N = N - 7
5     L = L + 1
      IF (L - 1000) 2, 7, 2
7     STOP
      END
      SUBROUTINE GENER (N)
      N = N + 1
      J = N ** 5
      N = J ** 5
      I = N - 31277
      J = N + 46777
      N = I - J
      END
      END
```

Go To 115

Matrix "HENRY" is placed in COMMON as follows

$$\begin{array}{llll}
H_{1,1} & H_{1,2} & H_{1,3} \ldots\ldots\ldots\ldots\ldots & H_{1,50} \\
H_{2,1} & H_{2,2} & H_{2,3} \ldots\ldots\ldots\ldots\ldots & H_{2,50} \\
\vdots & \vdots & \vdots & \vdots \\
H_{50,1} & H_{50,2} & H_{50,3} \ldots\ldots\ldots\ldots\ldots & H_{50,50}
\end{array}$$

Write a Fortran program which will transpose "HENRY" to "GENTRY" -- also in COMMON. "GENTRY" will be:

$$\begin{array}{lll}
H_{1,1} & H_{2,1} \ldots\ldots\ldots\ldots\ldots\ldots\ldots & H_{50,1} \\
H_{1,2} & H_{2,2} \ldots\ldots\ldots\ldots\ldots\ldots\ldots & H_{50,2} \\
\vdots & \vdots & \vdots \\
H_{1,50} & H_{2,50} \ldots\ldots\ldots\ldots\ldots\ldots\ldots & H_{50,50}
\end{array}$$

As you can see transposition involves changing rows to columns and columns to rows.

Go To 116

```
      PROGRAM TRANS
      DIMENSION HENRY (50, 50), GENTRY (50, 50)
      COMMON HENRY, GENTRY
      DO 10 I = 1, 50
      DO 10 J = 1, 50
   10 GENTRY (I, J) = HENRY (J, I)
      END
      END
```

Statement 10 could have been written as follows:

```
10 GENTRY (J, I) = HENRY (I, J)
```

Arrays are stored in consecutive locations. In the case of multi-dimension arrays, keep in mind, that the first subscript is advanced first while the others remain constant, etc. Thus the array A(3, 3) would be stored as follows:

A_{11}, A_{21}, A_{31}, A_{12}, A_{22}, A_{32}, A_{13}, A_{23}, A_{33}

Go To 117

Write Fortran statements which will provide for the following relationships:

a) $SUM = \sum_{k=10}^{20} (j_k^2 + 5)^k$

b) $d = 1/2 \sqrt{4R^2 - T^2}$, where $R = \cos(\theta/2)$

c) $Segment = R^2 \cos^{-1} \frac{(R - h)}{R} - (R - h)\sqrt{2Rh - h^2}$

d) $\text{Length of arc} = G\left[1 + \frac{2}{3}\left(\frac{2d}{e}\right)^2 - \frac{2}{5}\left(\frac{2d}{e}\right)^4 + \frac{2}{7}\left(\frac{2d}{e}\right)^6\right]$

e) Find the angle, θ, between two lines whose direction angles are α_1, α_2; β_1, β_2, and γ_1, γ_2, and given by:

$$\cos\theta = \cos\alpha_1 \cos\alpha_2 + \cos\beta_1 \cos\beta_2 + \cos\gamma_1 \cos\gamma_2;$$

f) $FINT = \frac{2(5a^3 x^3 - 6a^2 bx^2 + 8a b^2 x - 16b^3)}{35 a^4} \sqrt{ax + b} \tan x$

Go To 118

```
(A)    SUM = 0.
       DO 10 K = 10, 20
    10 SUM = SUM + (J(K)**2 + 5)**K

(B)    D = SQRT (4.*(COSF(THETA/2.))**2- T ** 2)/2.

(C)    SEGMENT = R**2*ACOSF((R- H)/R) - (R - H)*SQRTF(2.*R*H - H**2)

(D)    T = 2.*D/F
       ARCL = G*(1. + 2./3.*T**2 - 2./5.*T**4 + 2./7.*T**6)

(E)    THETA = ACOSF(COSF(ALPHA(1))*COSF(ALPHA(2)) + COSF(BETA(1))*COSF(BETA(2))
      1+ COSF(GAMMA(1))*COSF(GAMMA(2)))

(F)    FINT = (2.*(5.*(A**3)*(X**3) -6.*(A**2)*B*(X**2) + 8.*A*(B**2)*X
      1-16.*(B**3))*SQRTF(A*X+B)*TANF(X))/(35.*A**4)
```

Note continuations (digit 1 in column 6) of (E) and (F).
If you had several differences from the above -- review cards 2-25 concerning formula statements and symbols, otherwise

Go To 119

The following questions are related to CARD FORMAT.

a. In what columns must Fortran statements be written?

b. Can a Fortran statement start in column 10?

c. How is a "Comments" statement indicated?

d. How is a "continuation" indicated?

e. Can one save cards by putting two or three short statements on a card?

f. Can one statement of 670 characters be entered?

g. What should Card 1 of the program look like?

h. How can columns 73 through 80 be used?

i. What should the last card of a program look like?

j. How many columns are required for the following: TABLE=(3**2(M+N-6)+K)*4

k. Does the order of cards (except for first and last cards) make any difference in the program?

l. Since a key punch operator will probably punch your cards for you, can you suggest a way by which she can distinguish your numeric "O" from the letter "O"? (Kathy requested this question)

Go To 120

a. Columns 7 through 72

b. Yes-blanks are ignored (but a statement can't start below column 7)

c. "C" in column one

d. Column six is not blank or zero -- it contains a digit 1 to 9

e. No-only one statement per card is allowed.

f. No-the maximum number of characters per statement is 660.--there is a maximum of 66 statement characters per card and a maximum number of 10 cards (Initial statement card plus 9 continuation cards) per statement.

g. The first card of a program must be an identifying program name card such as:

```
            PROGRAM EXAMPLE
                ↑       ↑
Required Tag────┘       └─────Program name(Up to 7 characters)
```

h. Columns 73-80 are generally used to serialize the cards, but may be used for comments if desired. They are ignored during compilation.

i. The last card of a program must be an "END" statement starting in column 7.

j. 23 columns or columns 7-29 (if no spaces are used between characters)

k. Yes-order of the cards is crucial since control is sequential from one statement to the next.

l. A letter "O" is written Ø and a zero numeric, 0, is written 0.

Go To 121

Array, NUMBERS, contains 100 positive FIXED POINT values. Some of these are large values. Some are relatively small. It is desired to find the "factorial" of these values. Thus "factorial 5" (often written 5!) is (1)(2)(3)(4)(5) = 120. The programmer first decides to compare each term of "NUMBERS" with the constant, "KONST". If equal to or less than "KONST", the factorial is found by successive miltiplication, as in the above example. If the array term is greater than "KONST", the following rule is used.

$$\log n! = (n + \tfrac{1}{2}) \log n - n \log e + \log \sqrt{2n}$$

where π = 3.14159, e = 2.71828, "log is base 10", and n is the "NUMBERS" term.

HINT: If $\log n = J$, then $n = 10^J$

Write program, TORIAL, which will find and store factorials for the array, NUMBERS.

Go To 122

```
      PROGRAM TORIAL
      DIMENSION NUMBERS (100)
      DO 50 I = 1, 100
      IF (NUMBERS(I)) 6, 5, 6
    5 NUMBERS (I) = 1
      GO TO 50
    6 IF (NUMBERS(I) - KONST) 10, 10, 20
   10 N = NUMBERS(I)
   12 N = N - 1
      IF (N - 1) 50, 50, 15
   15 NUMBERS(I) = NUMBERS(I) * N
      GO TO 12
   20 FN = NUMBERS (I)
      F = (FN + .5)*LOG10F(FN) - FN*LOG10F(2.71828) + LOG10F(SQRTF(2.*3.14159))
      FN = 10. ** F
      NUMBERS(I) = FN
   50 CONTINUE
      END
      END
```

Some variations are possible. The first IF statement provides for factorial zero (=1). Also, formula calculations must be done in floating point since arguments for the common log and square root subroutines (from Reference Library), and constants π and e, are all floating point. The statement preceeding 50 then changes the floating point back to fixed point.

Go To 123

Havana Cuba: (A & P release)

Fido Castro made public a new computer today before a teeming mass at Della Rosa square.

Experts say a new saga of computer technology began with his announcement.

Although complete details are hazy, it is reported that Castro has ordered human calculating prodigies to be entombed as the inner working parts of the machine.

Castro ended his speech by claiming a real break through -- the elimination of programmer (bottle) necks!

The following example terminates this review. We hope you have continually reviewed past cards and the regular Fortran programming manual when necessary.

The 15 Puzzle

The "15 Puzzle" consists of numbers from 1 to 15 arranged in a square array, as FIG. 1. The only rule is: "Any one of the numbers to the immediate right, left, top, or bottom of the blank space can be moved to the empty space. In FIG. 1, only 15 or 12 can be moved. The puzzle is: "given some array (for example FIG. 2), can the numbers in FIG. 1 be moved (by the rule) so as to achieve this new array".

FIG. 1

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

Mathematically it can be shown that there are 16 factorial different arrays possible. Exactly one-half (over 10 trillion) arrays are impossible to ever achieve. After this puzzle was introduced (about 1870) to the public it became very popular. So much time was spent on it, that in France, a law was passed prohibiting "puzzle 15" attempts during working hours. Huge sums of money were won by clever people who bet others (less clever) that certain arrays (which were impossible to achieve) could not be achieved.

FIG. 2

1	2	3	4
5	6	7	8
9	10	11	12
13	15		14

Then someone came up with a method which indicated whether or not a specific array might be achieved. This method is described on the next card.

Go To 124

(1) Starting Array

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

(2) "To Be Achieved Array"

1	2	3	4
5	6	7	8
9	10	11	12
13	15		14

(3) Lettered Positions

A	B	C	D
E	F	G	H
I	J	K	L
M	N	O	P

A neat trick to determine if a particular array can be achieved is as follows:

a. Assume individual squares are positioned by letters shown in (3).

b. Consider the number (call it "X") in position A, of "the array to be achieved". Count how many numbers smaller than "X" are in positions higher lettered than A. For (2), this would be zero (count the blank as 16).

c. Do this for all positions, and add up the counts thus obtained for positions A through P. (Total count for (2) is 1)

d. If the blank square is one of B, D, E, G, J, L, M, or O, add 1 to the sum otherwise leave sum alone.

e. If sum is even, the array can be achieved. If sum is odd, it can not.

Assuming a certain "to be achieved array" "MAYBE" is stored in COMMON; write a Fortran program which will follow steps "a" through "e" above to test if achievement is possible. If it can be achieved, store a 1 at "ANSWER". If it can't, store a 2.

Go To 125

```
      PROGRAM PUZZLE
      DIMENSION MAYBE(4, 4)
      COMMON MAYBE
      KOUNT = 0
      IF (MAYBE(1, 2) - 16) 1, 8, 1
    1 IF (MAYBE(1, 4) - 16) 2, 8, 2
    2 IF (MAYBE(2, 1) - 16) 3, 8, 3
    3 IF (MAYBE(2, 3) - 16) 4, 8, 4
    4 IF (MAYBE(3, 2) - 16) 5, 8, 5
    5 IF (MAYBE(3, 4) - 16) 6, 8, 6
    6 IF (MAYBE(4, 1) - 16) 7, 8, 7
    7 IF (MAYBE(4, 3) - 16) 9, 8, 9
    8 KOUNT = KOUNT + 1
    9 DO 30 I = 1, 4
      DO 30 J = 1, 4
      DO 20 K = 1, 4
      DO 20 L = 1, 4
      IF (MAYBE(I, J) - MAYBE(K, L)) 20,20,10
   10 KOUNT = KOUNT + 1
   20 CONTINUE
   30 MAYBE (I, J) = 16
      N = KOUNT/2
      IF (KOUNT - 2*N) 50, 40, 50
   40 ANSWER = 1.
      GO TO 60
   50 ANSWER = 2.
   60 END
      END
```

Chances are your solution will not be the same as the above, but perhaps it does provide an indication of the correctness of your solution. Note how the even or odd test was made. The remainder is lost on integer division, and thus, an even multiple of 2 test can be used to determine if the number is even.

Go To 126

The logical flow of most problems can be shown by the above diagram. Up to this point, this course has presented the FILTERING characteristics of Fortran Programs. The final section of the course will be devoted to the INPUT - OUTPUT characteristics.

Fortran provides for moving information to or from computer storage by special and unique input-output statements. Two types of statements are all that is needed for the movement of data! One type indicates the medium or device that is to do the moving and at the same time, lists the data to be moved. The other type formats or describes (by letter codes) how the data is to be moved. These two types may be referred to as:

(1) Input-Output Data List Statements

(2) Format Statements

Go To 127

This Fortran course is oriented, for Input, toward cards and magnetic tape; for output, toward cards, magnetic tape, and printer. (Other media are not covered)

There are five different "Input-Output Data List Statements".

"READ" statement --- for INPUT by CARDS
"PUNCH" statement --- for OUTPUT by CARDS
"READ INPUT TAPE" statement --- for INPUT by MAGNETIC TAPE
"WRITE OUTPUT TAPE" statement --- for OUTPUT by MAGNETIC TAPE
"PRINT" statement --- for OUTPUT by PRINTER

The same organizational plan is used by all five to indicate what data is to be moved, and how it is to be moved. The example below indicates the method used:

```
READ  10, A, B, C, D
```

"DATA LIST" -- tells what data is to be moved--in this case, A, B, C, D

"STATEMENT NUMBER" -- tells where to find a FORMAT statement which in turn will describe how to move the data. In this example, the FORMAT statement is located at statement 10 in the Fortran Program.

"READ" indicates INPUT by CARDS

Go To 128

Each "Input-Output Data List" statement (of which there are five in this course) consists of three parts: (1) medium name, (2) statement number, (3) data list. These are re-emphasized in the following example.

READ INPUT TAPE 3, 12, A, B, C, D

data list

medium name = Magnetic Tape No. 3

statement number where FORMAT statement is located

Note that when Magnetic Tape is involved, a tape unit number must be present to indicate which tape is used. In writing programs for the Minneapolis 1604 Service Bureau, use tape units 2, 3, or 4; since tape unit 1 is reserved for the Reference Library.

Another example follows:

PRINT 12, XA, XB, XC

data list

Statement number where FORMAT statement is located

Medium name = printer

Go To 129

Of the three parts of the "Input-Output Data List" statement, the third (DATA LIST) involves some restrictions. Let's look at these.

1. Only variable quantities can appear--constants are not allowed. (See any example below.)
2. If individual quantities are to be moved, the variable names simply appear in the list separated by commas. (See Ex. 1 below)
3. If an entire array is to be moved, the name of the array can appear in the list as if it were a single variable. (See Ex. 2 below)
4. If only special members of an array are to be moved, or if a special order is described, a special notation similar to DO loops is used. (See Ex. 3 below)

```
Ex. 1     PRINT 16, X, NET, B
Ex. 2     PUNCH 202, MATRIX
Ex. 3     WRITE OUTPUT TAPE 4, 75, (DATA(I), I = 2, 60, 2)
```

Ex. 3 tells the Compiler to write on Magnetic Tape Unit No. 4, according to the FORMAT located at statement number 75. The array "DATA" is to be written starting with $DATA_2$, then $DATA_4$, etc. ..... to $DATA_{60}$. Also note comma required after DATA(I).

Go To 130

A subscripted variable in the data list is probably the most difficult part of writing "Input-Output Data List Statements". Another example re-emphasizes this aspect.

```
WRITE OUTPUT TAPE 4, 10, ((B(I,J), J = 2, 50, 2), I = 1, 50)
```

The above statement provides for data output on Magnetic Tape Unit 4, according to FORMAT statement 10. Thinking of the data list as nested DO Loops help. For example:

```
((B(I,J), J = 2, 50, 2), I = 1, 50)
  Inner Loop
              Outer Loop
```

The data would be written as follows

$B_{1,2}$, $B_{1,4}$ ..... $B_{1,50}$

$B_{2,2}$, $B_{2,4}$ ..... $B_{2,50}$ etc.

Note: An open parenthesis begins the loop and a closed parentheses followed by a comma must follow each loop. No comma is needed, however, after the last closed parentheses in the list. (The arrows above indicate parentheses and commas.)

Go To 131

We started the Input-Output section by emphasizing that two types of statements were involved. The first, which we called "Input-Output Data List Statements" indicate the medium to be used and what data is to be moved. The second type statement required is the "FORMAT STATEMENT". This statement permits the programmer to tell the Compiler how data is to be moved. Note, the FORMAT statement causes no action in itself. It is simply used as a reference for some "Input-Output Data List" statement. It tells how the characters (alphabetic or numeric), representing the values of the variables in the "Data List", are converted and positioned during Input or Output.

Every FORMAT statement includes a statement number, followed by the word "FORMAT", which in turn is followed by a FORMAT "specification" enclosed in parentheses. For example:

$$\texttt{10 FORMAT } (\underbrace{\texttt{I2, E12.4}}_{\text{"Specification"}})$$

The crux of the problem is to learn how to write the "specification"!

Go To 132

One writes format specifications by using <u>codes</u> made up of letters, numerics, and symbols. For this course, the following six code forms are recommended:

nIw	=	the number, n, of INTEGER FORM FIXED POINT fields of width, w. ("n" and "w" are decimal integers)
nEw.d	=	the number, n, of EXPONENTIAL FORM FLOATING POINT fields of width, w, including "d" significant digits to the right of the decimal point. ("n", "w", and "d" are decimal integers and "w" includes "d")
nX	=	the number, n, of blanks (or spaces) that are to be processed.
nH	=	the number, n, of HEADING (or Hollerith) characters to be processed.
/	=	"the slash". This symbol indicates that the <u>next</u> card or line should be read or written.
( )	=	"parentheses". These are used in special ways to indicate continued input-output control.

Study the above forms carefully, then

Go To 133

Let's start with code forms nIw and nEw.d -- which are defined by an example:

n = number of fields   w = <u>width</u> of each field   d = number of significant digits of width, w, which are to the right of decimal point

```
     READ  10  BETA,  JET
10   FORMAT  (6E9.3   5I4)
```

"READ" indicates INPUT by CARDS, "10" locates the FORMAT statement, "BETA" is a 6 member FLOATING POINT array, JET is a 5 member FIXED POINT array.

n = 5, indicates there are 5 fields or members
I = INTEGER, indicates INTEGRAL INPUT - OUTPUT (Input in this case)
w = 4, indicates each INTEGER is contained on the card in FIELDS of 4 COLUMNS each (4 columns = width)

n = 6, indicates there are 6 fields or members
E = "EXPONENTIAL NOTATION". - data is expressed with a sign ($\pm$), the decimal point, the significant digits, the letter "E", the sign of the exponent (+ or -), and 2 digit exponent.
w = 9, indicates each number is expressed in a field of 9 columns.
d = 3, indicates 3 significant digits <u>of</u> <u>the</u> <u>width</u> are to the right of the decimal point.

Go To 134

The programmer picks the codes to fit his programs--just as a "saw-bones" picks his medicines to fit his patients! For example:

On line 1, print a sum (less than 1000), now in KTRS
On line 2, print a product (less than 10,000), now in KTRP
On lines 3, 4, etc., print 100 terms of array, DATA; four numbers per line, each with six digits of significance to the right of the decimal point.

```
    PRINT 12, KTRS, KTRP, (DATA(J), J = 1, 100)
 12 FORMAT (I3,/, I4,/, (4(E12.6, 2X)/))
```

"I" is required for KTRS and KTRP since these are FIXED POINT variables. I3 provides for an INTEGER field width less than 1000; I4 for a field width less than 10,000.

Each of the first 2 slashes (/) causes the starting of a new line.

(4(E12.6, 2X)/) in conjunction with (DATA(J), J = 1, 100) does the following:

Prints four(4) FLOATING POINT results per line, in EXPONENTIAL (E) form (for example -- +.625732E+04) each in a field width twelve (12); of which six digits (6) are significant and to the right of the decimal point, each followed by two spaces (2X), until one hundred (J = 100) terms are printed.

Go To 135

We hope the previous example enabled you to deduce the following facts about "I" and "E".

1. FIXED POINT variables in the data list require the code form, nIw, in the corresponding position of the FORMAT specification. Likewise, FLOATING POINT variables in the data list require a code form, nEw.d, in the corresponding position of the FORMAT specification.

2. nIw in the FORMAT specification indicates that FIXED POINT INTEGERS (with no decimal point expressed) are involved. nEw.d in the FORMAT specification, indicates that EXPONENTIAL FORMS of FLOATING POINT numbers are involved.

   - . 1262 E + 03 is an example of an nEw.d number, where:

   1 → Two digits for exponent of base 10
   2 → Symbol for sign (+ or -) of exponent
   3 → Letter "E" meaning "exponent"
   → One digit for <u>each</u> significant digit
   4 → The decimal point
   5 → Symbol (+ or -) for sign of number

Thus - . 1262 E + 03 = 126.2

Note: To determine the width (w) of an nEw.d form, a total of 6 (see circled) positions) must be added to the number of significant digits desired. Thus if 8 significant digits (d) are required, the width (w) is 8 + 6 = 14 or E14.8.

Go To 136

Data appears on one INPUT card in the following field order. (Δ = space)

Card Field Number	Real Value		Card Form	Card Field Number	Real Value		Card Form
1	5	=	ΔΔΔ5	5	612.7	=	+.6127E + 03
2	120	=	Δ120	6	-.03172	=	-.3172E - 01
3	326	=	Δ326	7	512.1	=	+.5121E + 03
4	64	=	ΔΔ64	8	-7.125	=	-.7125E + 01

One programmer wrote the following:

```
      READ 21, IOTA, JET, K, L,  (ALPHA(I), I = 1, 4)
21    FORMAT (I4, 2I4, I4, 4E10.4)
```

Another programmer wrote the following:

```
      READ 312, NNN, MIN, J, NN, (B(I), I = 1, 4)
312   FORMAT (4I4, 4E10.4)
```

If you think both solutions are correct, go to 138.
If you think only one is correct, go to 137.
If you think none are correct, go to bed earlier tonight, and start at 133 tomorrow.

Actually the two sets of statements given were both correct. The only difference being in the naming of variables and the manner in which the part of the FORMAT specification corresponding to the FIXED POINT data was written.

Thus FORMAT (I4, 2I4, I4) is identical in meaning to FORMAT (4I4). The first, tells the Compiler to read 1 field of 4 columns, then 2 more fields of 4 columns each, and finally 1 field of 4 columns -- a total of 16 columns. The second, tells the Compiler to read 4 fields of 4 columns each -- again a total of 16 columns.

Either set of variable names used was correct since it makes no difference how a variable is named as long as the mode of the name fits the mode of the data (FIXED or FLOATING), and the name is not longer than seven characters.

Go To 138

The next example uses the specification characters: "X", "H" and "/" (slash). Mean, M, and sigma, IGMA, (both integers), and 100 random numbers, are computed. Each random number of array, RANDOM, contains FLOATING POINT numbers of eight significant digits. Write the OUTPUT statements which will list the output on Magnetic Tape Unit 4 in a form <u>similar</u> to the following (Δ = space) (_ = an output digit)

```
MEANΔANDΔSIGMA = ___ΔΔΔ__                             (Line 1)

4 random numbers per line separated }                 (Lines 2,3,4, ... to 26)
by 5 spaces each.                   }
```

```
      WRITE OUTPUT TAPE 4, 17, M, IGMA, (RANDOM(J), J = 1, 100)
  17  FORMAT (15HMEANΔANDΔSIGMA= I3,   3X   I2,   /,   (4(E14.8, 5X))
```

Write 15 Hollerith characters: "MEANΔANDΔSIGMA ="

Produces 3 digits for MEAN

Provides for 3 spaces

Produces 2 digits for IGMA

Starts new line

Provides for 4 EXPONENTIAL form numbers per line, separated by 5 spaces each, until j = 100

Go To 139

From the previous example, one can see that codes: X, H, and / (slash) are easily learned.

X is used to skip or provide spaces within a line or card.

35X on INPUT means: skip 35 columns.

35X on OUTPUT means: skip 35 spaces.

/ (slash) is used to start a new line or card.

/// on INPUT means: start READING the third card or line.

/// on OUTPUT means: start PRINTING, WRITING, or PUNCHING the third line or card.

Note that a slash (/) performs the same function as a carriage return on a typewriter.

H is used to provide headings (hollerith) characters either on OUTPUT or within a FORMAT statement on INPUT.

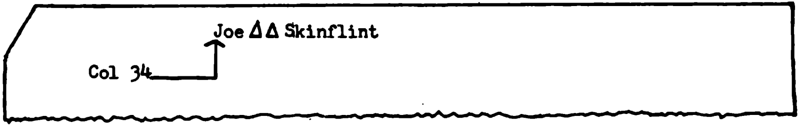
7HMINNY=5 on OUTPUT gives: MINNY=5

7HMINNY=5 on INPUT substitutes the next 7 characters read in during INPUT for the 7 characters, MINNY=5, in the FORMAT statement.

Go to 140

Joe Skinflint was known as a "tightwad". One night when the computer center was empty (just before a holiday), he sneaked into the center and ran the program described below.

Program, MOOCHER, produces "calling cards" for Joe by punching his name on each of 500 cards. Each card will appear as follows, starting at column 34. (Δ = space)



Write the program, MOOCHER, and don't worry about the punched cards being <u>interpreted</u>---that's Joe's problem.

Go To 141

```
    PROGRAM MOOCHER
    DO 5 I = 1, 500
 5  PUNCH 10
10  FORMAT (33X, 14HJOEΔΔSKINFLINT)
```

Provides for punching 14 Hollerith characters: JOEΔΔSKINFLINT

Provides for 33 spaces before punching starts in column 34

Notice that the data list statement, PUNCH 10, must be placed in a DO LOOP since the data list has no variables. In other words, one can not write: "PUNCH 10, I = 1, 500" since no subscripted variable is possible for "I = 1,500" to operate on.

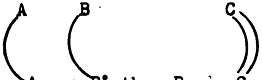
Go To 142

The most difficult aspect of writing FORMAT specifications is the correct use of parentheses for subscripted variables. We have been using some of these concepts in past examples, but more details are now given.

Three basic "parenthesis structures" are important.

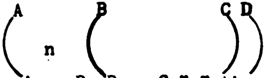
A ⟶ B; repeat to end of list

Example: (E14.8). Process left to right, to end of list. Each time closed parentheses is reached, a new line of output is started.



A ⟶ B; then B ⟶ C repeatedly to end of list

Example: (I3,(E6.2)). Process left to right to inner most parentheses. Remain within the inner most parentheses to end of list. Each time inner most closed parentheses is reached, a new line output begins, starting again at the inner most open parentheses.



A ⟶ B, B ⟶ C "n" times, C ⟶ D, repeat from A if list is not exhausted.

Example: (2I5,6(E14.8),/). Process from left to right to first inner parentheses. Remain within the inner parentheses for 6 lines; then continue processing to the next closed parentheses. If data list is not exhausted, start again from first open parentheses on a new line of output.

Go To 143

(from the Los Angeles Post)

Wanted: programmer for new IRC computer system. must be experienced with other systems. will give the right man a free draining course which will enable the IRC to be programmed. for right man willing to dry up salary $25000 after we drain you.

Mrs. J. C. Prim supervised the programming section of the Outer Banks project. Known as a tireless and skilled FORTRAN "pusher", she insisted upon the utmost efficiency from her section programmers (slaves).

Her annoyance with Herb Waster (who had often previously wasted her time) reached a breaking point when she examined the following portion of a program he had written.

```
       WRITE OUTPUT TAPE 3, 8 (BETA(I), I = 1,5)
     8 FORMAT (E14.8)
       .        .        .        .
       .        .        .        .
       PRINT 22, (ALPHA(I), I = 1,5)
    22 FORMAT (5(E14.8))
       .        .        .        .
       .        .        .        .
```

"Can't you see", she snapped at Herb, "that both format statements do exactly the same thing -- and consequently one can be eliminated entirely from the program".

"And don't argue with me", she continued, as Herb started to protest, "or I'll have you terminated immediately".

What do you think? Was Mrs. Prim right?

If you think she was, Go to 144.
If you think she was not, Go to 145.

Here lies the soul of Janella Prim
Alas, for her, life became too grim,
She withstood slander, rumors, and blame
But Herb's "double format" ended the flame.

There was a difference!

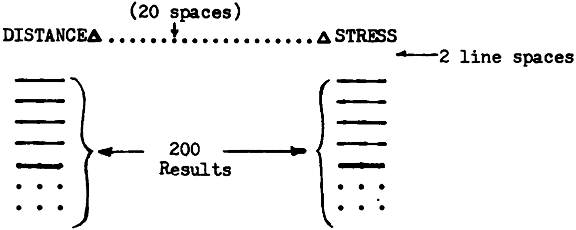
(E14.8) provides for <u>one</u> number <u>per</u> <u>line.</u>

(5(E14.8)) provides for <u>five</u> numbers on <u>one</u> line.

Go To 145

Now that Mrs. Prim met her "Wasterloo", let's try the following:

An engineer wants to PRINT the OUTPUT of one of his problems in a form similar to the following: (Δ = space, —— = output digits)



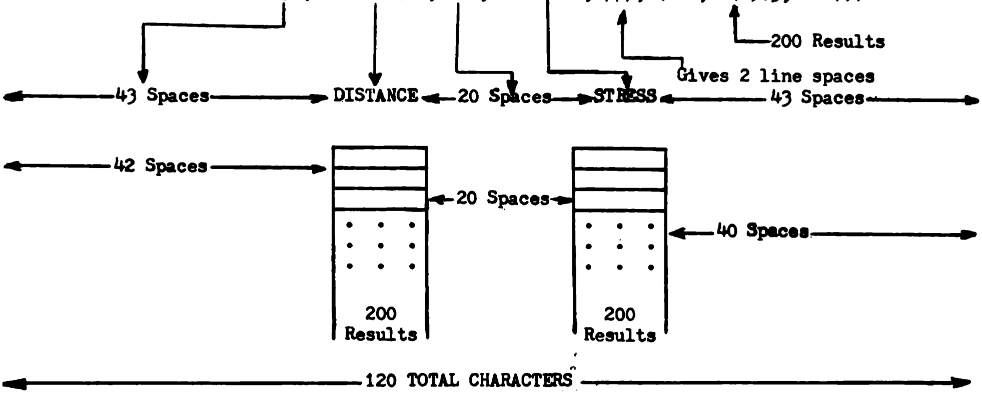
DISTANCE and STRESS are headings (Hollerith) for computed results. The 200 results are each computed to 3 significant figures. 2 line spaces exist between headings and first lines of computed results.

Write OUTPUT statements which will provide for PRINTED pages in a form similar to the above; then

Go To 146

The two OUTPUT statements, referenced to the OUTPUT is shown below:

```
PRINT 63, (DIST(I),  STRESS(I), I = 1, 200)
63 FORMAT (43X, 8HDISTANCE, 20X, 6HSTRESS, ///, (42X, 2(E9.3, 20X)))
```



Go ™- 147

"Systy" and "Cysty" were identical twins. In physical appearance both were alike but in work habits they differed markedly. For example, "Systy" once wrote a program using the following statements:

```
   READ 10, JET, (ALPHA(I), I = 1,100)
10 FORMAT (I4, (2E16. 4)
```

"I'm having trouble", said Systy. "I try to read in the number 123 and I get 1230, and I can't compile the FORMAT statement. That computer isn't working right".

"It's not the computer", said Cysty. "First, 1230 came in because you wrote 123 as 123▲ on the card. You did not RIGHT ADJUST your input. As for the FORMAT error, you forgot a very important rule. The <u>number of open parentheses</u> must equal the <u>number of closed parentheses</u>--you have two opens and only one closed".

"All I hear are rules", moaned Systy, "don't you know rules are made to be broken".

"If that is true", replied Cysty, "then compiler programs are written <u>not</u> to run, because the "break a rule Compiler "has yet to be born".

Go To 148

Programs which do not provide for changing INPUT or OUTPUT are restrictive. For example, a program written to READ in exactly 100 data cards is not applicable for 101 cards, etc. To attain flexibility, one often provides for the first card of a "data batch" to be a <u>control</u> or <u>header</u> card. This card then regulates control of the data that follow. The following example emphasizes this technique.

An array, FILE, is READ in from cards. Card format is:

Card 1:	<u>Number</u> of cards, N, in the FILE. Assume N can range from 1 up to 10,000.
Cards 2,3,4 --- to N:	One number per card ranging from -.999 to +.999

Input statements which will provide for READING one FILE at a time are:

```
      READ 20, N, (FILE(I), I = 1, N)
   20 FORMAT (I4,/, (E9.3))
```

Provides for READING a value (up to 4 digits) for N, and then starting to READ the next card

Provides for READING 1 number per card until N cards are READ

Go To 149

ROTATE THIS CARD 90 DEGREES FOR A SURPRISE QUIZ

Answer The Following

1. "nIw" in a FORMAT statement describes what type of variable in the related data list?

2. To represent "pi", would one probably use nIw or nEw.d in the FORMAT statement?

3. BETA, holds the OUTPUT. However, one wants to use nIw in the FORMAT statement. What correspondingly must be used in the data list?

4.
```
    PRINT 20
 20 FORMAT (6HTABLES)
```
Will produce what?

5. Fortran statements can only use card columns 7 through 72. Is this also true with respect to INPUT/OUTPUT data?

6. (3I2, (E12.6)) in a FORMAT statement infers repeated patterns of 3 integers and 1 floating point form number until the list is ended. (True or False)

7. -.00612 E + 05 equals what number? Write an nEw.d for 6 significant digits to the right of the decimal point.

8. Since the 1604 is a binary computer, is it necessary for the programmer to convert decimal numbers to binary before using them?

9. To skip 8 spaces one can use "8X"; to space to the fifth line, use "5/". (True or False)

10. Is there anything amiss in a FORMAT specification such as (50X, 12E6.2)?

11. If two digit numbers are to be READ in through a field width of 4, can the numbers appear in any of the four columns of the field?

12. Given:
```
    READ 10, JETTY
 10 FORMAT (5HJONES, I3)
```
The first 8 columns of the card contain SAP=2177. What action takes place?

Go To 150

1. Format code form, nIw, means INTEGER FIXED POINT. Therefore, the data list variable must be a FIXED POINT variable; it must begin with I, J, K, L, M, or N.

2. Format code form, nEw.d would probably be used since most calculations involving pi would want to carry some digits to the right of the decimal point in pi.

3. The array name would have to be changed to FIXED POINT - for example, JBETA.

4. One line of heading, TABLES, will be printed starting at the left margin.

5. No. INPUT data cards can use the whole card width of 80 columns. All other INPUT - OUTPUT media use 120 characters.

6. False. The pattern inferred is 3 fixed point numbers followed by <u>repeated</u> floating point numbers until the list is exhausted. The 3 fixed numbers are not repeated.

7. -.00612E + 05 = -612. For 6 significant digits, the form would be: E12.6

8. No. All numbers are written in decimal-the compiler does the conversion for you.

9. False. 8X provides 8 spaces, spacing to the fifth line is: /////. (5 slashes)

10. Yes. "50X" requires 50 columns, "12E6.2" requires 72 columns. The maximum number of columns allowed is 120 for magnetic tapes and line printer, and 80 for cards.

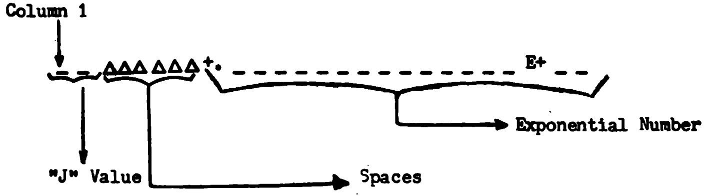
11. No. Input data must be RIGHT ADJUSTED within the field. Thus, Δ23Δ will equal 230, not 23.

12. SAP-2 (first 5 characters) will replace JONES in the format statement. The next 3 characters (177) will be assigned to the variable "JETTY".

Go To 151

Program, MISFIT, was having trouble! Date coming in was incorrect. In checking the card format against the Input statements, the following correspondence was found:

CARD FORMAT



CORRESPONDING INPUT STATEMENTS

```
     READ 35, J, F(J)
35   FORMAT (I3, 5X, E16.10)
```

Can you see any trouble spots in the above. If not, go to card 153. If you do see "trouble spots".

Go To 152

You were right, there were two trouble spots. First, the FORMAT statement called for "I3" (which infers 3 columns for width) and only 2 columns were used--the number of the subscript was not RIGHT ADJUSTED. Second, there were 16 columns provided for an E field of 20 columns. This means the exponent value would be lost in reading in.

A 30 by 30 matrix, called K, is ready for output on the PRINTER. However, only the principal diagonal members are to be printed in the following form:

```
K(1,1)=---
K(2,2)=---   etc.
```

The output statements follow. DONT'T PEEK--- try writing them first.

```
   PRINT 27, (N, N, K(N, N), N = 1,30)
27 FORMAT (2HK(,I2, 1H,, I2, 2H)=, I3)
```

Study the above until you understand the solution! Note, the variable in the PRINT statement could have been any FIXED POINT variable (for example, "I", "J", etc.)

Go To 154

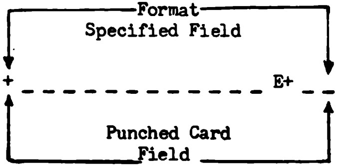
According to the input statements there were two "trouble spots" on the card.

(1) The "I" field was not right adjusted--it was not 3 columns wide as required in the format statement. Thus the subscript read would be 10 times larger than intended.

(2) The "E" field punched on the card was in 20 columns instead of the 16 columns specified in the format statement and therefore the exponent would not have been recognized as part of this number.

REMEMBER---

The field specified in the format must be in <u>exact</u> agreement with the input card field, as the example below:



If you now understand the above "trouble spots", you are ready for the next problem on 152.

At one time or another, we all try to reach "the end of the rainbow". And, more often than not, the pot of gold we visualize at its end, seems ever just beyond our outstretched fingers.

Perhaps part of the trouble lies in our inherent belief that all things must end--when in fact, some may not have an ending.

Paradoxically, as one appears to approach his various "endings", there is a natural desire to look back--to retrospect--and to trace the journey from the beginning. Thus the ending flows into the beginning and one finds himself looking forward to a new ending.

In this course of study, we have reached such an ending--perhaps only a milestone on the journey toward complete understanding of the concepts already presented. Nevertheless, it appears to be a point where one can pause, and looking back, ask himself, "have I established a firm foundation of understanding these programming concepts?"

With this question in mind, we hope you will conclude this course by an all out attempt to solve the following ten problems. Let them be a barometer by which you can judge how close you might be to this course "ending".

Go To 155

ONE coming up, NINE to go
Knock this off and start the show!

Write a Fortran program for the following hypotheses:

INPUT -- Cards; three arrays, X, Y, and IOTA. X and Y each contain 1000 items ranging from -2.510 through +2.499. IOTA contains 1000 integers ranging from 100 to 2000. Each card has the format:

IOTA(I), 2 spaces, X(I), 2 spaces, Y(I)

BODY OF PROBLEM --- Form the following AFTER INPUT:

$$\boxed{C_i = X_i^3 + Y_i^3} \quad \text{where } i = 1, 2, 3 \ldots, 1000$$

If $C_i \geq 0$, Increase C_i by $IOTA_i$

If $C_i < 0$, Increase counter, KTR, by 1

OUTPUT -- By PRINTER. On line 1, (centered), print KTR= and follow this with the accumulated count in KTR. Then line space the next 2 lines. Starting on the fourth line, print 4 items of C per line, each of 10 significant digits, with 5 spaces between items.

Go To 156

```
      PROGRAM ONE
      DIMENSION X(1000), Y(1000), IOTA(1000), C(1000)
      KTR = 0
      READ 10, (IOTA(I), X(I), Y(I), I = 1, 1000)
   10 FORMAT(I4, 2(2X, E10.4))
      DO 40 I = 1, 1000
      C(I) = X(I) ** 3 + Y (I) ** 3
      IF(C(I)) 20, 30,30
   20 KTR = KTR + 1
      Go To 40
   30 FIOTA = IOTA(I)
      C(I) = C(I) + FIOTA
   40 CONTINUE
      PRINT 50, KTR, (C(I), I = 1, 1000)
   50 FORMAT (56X, 4HKTR=,I4,///,(4(E16.10, 5X)))
      END
      END
```

Note:

1) The inclusion of array C in the DIMENSION, since 1000 C values were computed.

2) FIXED POINT IOTA(I) is floated (statement 30) before its use in a FLOATING expression.

3) The statement KTR = 0 could have been eliminated since variables are all set to zero initially by the compiler.

Go To 157

TWO coming up, EIGHT to go
Do this too, see confidence grow!

INPUT -- NONE

BODY OF PROBLEM -- Starting with $k = 1$ and $S = 0$, and successively increasing k by 1 find successive values of S, (by adding each new S to the preceding S value) until S is equivalent to or greater than 0.5772157, where:

$$S = \sum_{1}^{n} e^{-k} \log k \text{ (natural logarithm)}$$

OUTPUT -- PRINT out on line 1, starting at the left hand column, the following:

NUMBER OF TERMS NEEDED = _ _ _

where _ _ _ contains the last value of k used

Remember: e^x and log x are available on REFERENCE LIBRARY.

Go To 158

```
      PROGRAM TWO
      FK = 1.
      S = 0.
 10   S = S + EXPF (-FK) * LOCF(FK)
      If (S -.5772157) 20, 30,30
 20   FK = FK + 1.
      Go To 10
 30   K = FK
      PRINT 40, K
 40   FORMAT (25HNUMBER▲OF ▲TERMS▲ NEEDED△=△, I3)
      END
      END
```

Note:

The arguments for Reference Library subroutines EXPF and LOGF must be FLOATING POINT; therefore, you may not write EXPF (-K) and LOGF(K). Thus FK is used instead of K.

However, the PRINTED value of K is desired in FIXED POINT; therefore, it is necessary to "re-fix" the floated FK so that it returns to FIXED POINT (see statement 30).

Go To 159

THREE coming up, SEVEN to go
Don't give up, know, know, know!

INPUT -- The corresponding slopes of paired lines are in arrays "S" and "T". The number of pairs is 1000, and slopes range from -.99 through +.99. This data is recorded on Tape Unit 3, where each record lists an "S" slope, 2 spaces, and a "T" slope.

BODY OF PROBLEM -----

FIND $\begin{cases} X_i = 2 \sin \beta_i \\ Y_i = 2 \cos \beta_i \\ Z_i = 2/\beta_i \end{cases}$ where $\begin{cases} \beta_i = \dfrac{\cos \theta + 1 + S_i}{1 + T_i} \\ \theta = 0.125\ i \text{ radians} \\ S_i \text{ and } T_i \text{ are corresponding slopes} \end{cases}$

β_i, the ANGLE above, is to be written as a SUBROUTINE, using COMMON storage for arrays, S and T.

OUTPUT -- On Tape Unit 4, list on each line, Xi, Yi, and Zi, each to 8 significant digits, separated by 5 spaces. Following output, tape 4 is to be rewound to its starting point.

Go To 160

```
      PROGRAM THREE
      DIMENSION S(1000), T(1000), X(1000), Y(1000), Z(1000)
      COMMON S, T
      READ INPUT TAPE 3, 10, (S(I), T(I), I = 1, 1000)
   10 FORMAT (E8.2, 2X, E8.2)
      DO 20 I = 1, 1000
      CALL ANGLE (I, B)
      X(I) = 2. * SINF(B)
      Y(I) = 2. * COSF(B)
   20 Z(I) = 2./B
      WRITE OUTPUT TAPE 4, 30, (X(I), Y(I), Z(I), I = 1, 1000)
   30 FORMAT (3(E14.8, 5X))
      REWIND 4
      END
      SUBROUTINE ANGLE (J, BETA)
      DIMENSION S(1000), T(1000)
      COMMON S, T
      FJ = J
      BETA = (COSF(.125 * FJ) + 1. + S(J))/(1. + T(J))
      END
      END
```

Note

(1) The exact correspondence between the modes of the parameters of the CALL statement and the dummy variables of the SUBROUTINE statement.

(2) The argument for the COSF subroutine could have been COSF(THETA) if the statement THETA = FJ * 0.125 had been included.

FOUR coming up, SIX to go
Hang on now or stub your toe!

The radius of gyration, R, is given by the rule:

$$R^2 = \frac{m_1 r_1^2 + m_2 r_2^2 + \text{-----} \; m_n r_n^2}{m_1 + m_2 + \text{-----} \; m_n} \quad \text{where} \quad \begin{array}{l} m = \text{mass} \\ r = \text{radius} \end{array}$$

Groups of masses and corresponding radii (each of 10 significant digits) are READ in as a job group. Preceding each job group is a control card containing the number of cards, N, in that group, and the job group number, NJOB. Control card with N = 0, indicates that all data has been READ in. Maximum number of cards per job group is 1000 and NJOB numbers can range from 1 to 9999.

Write a program which will READ in each job group, calculate R (by above rule), and write results on Tape Unit 4, on <u>one</u> line of OUTPUT as follows: (Δ = space)

NJOBΔ=Δ_____ΔΔΔΔ NΔ=Δ_____RADIUSΔOFΔGYRATION=______

INPUT cards have the following format:

Control card: N, 2 spaces, NJOB

Other cards: mass, 2 spaces, radius

Go To 162

```
      PROGRAM FOUR
      DIMENSION FM(1000), R(1000)
    1 READ 2, N,NJOB
    2 FORMAT (I4, 2X, I4)
      IF(N) 5, 200, 5
    5 READ 7, (FM(I), R(I), I = 1, N)
    7 FORMAT (E16.10, 2X, E16.10)
      SUMMASS = 0.
      RA = 0.
      DO 10 I = 1, N
      RA = RA + FM(I) * R(I) **2
   10 SUMMASS = SUMMASS + FM(I)
      RA = SQRTF (RA/SUMMASS)
      WRITE OUTPUT TAPE 4, 100, NJOB, N, RA
      Go To 1
  100 FORMAT (7HNJOBΔ=Δ,I4, 4X, 4HNΔ=Δ,I4, 19HRADIUSΔORΔGYRATION=,E16.10)
  200 END
      END
```

Go To 163

FIVE coming up, FIVE to go
Ask for trouble, you sure get moah!

Given a 3 x 3 matrix, A_{rc}. The problem is to develop the derived matrix, B_{rc}, using the rules: If row number, r, of B_{rc} is <u>equal</u> to its column number, c, then use:

$$B_{rc} = A_{rc} - \sum_{k=1}^{k=(r-1)} (A_{rk})(A_{kr}) \quad \text{, if } k < 1, \text{ summation} = 0$$

If row number, r, of B_{rc} is <u>larger</u> than its column number, c, use:

$$B_{rc} = A_{rc} - \sum_{k=1}^{k=(c-1)} (A_{rk})(A_{kc}) \quad \text{, if } k < 1, \text{ summation} = 0$$

If row number, r, of B_{rc} is <u>smaller</u> than its column number, c, use:

$$B_{rc} = \left[A_{rc} - \sum_{k=1}^{k=(r-1)} (A_{rk})(A_{kc}) \right] \left(\frac{1}{A_{rr}} \right) , \quad \text{if } k < 1, \text{ summation} = 0$$

A_{rc} with nine elements ranging from 00.1 to 99.9, is READ in row by row; where one element is on each card. Compute the derived matrix, B_{rc}, by the above rules, and PRINT out the derived elements in the form:

B(1,1)Δ=______ B(1,2)Δ=______ B(1,3)Δ=______

B(2,1)Δ=______ B(2,2)Δ=______ B(2,3)Δ=______

B(3,1)Δ=______ B(3,2)Δ=______ B(3,3)Δ=______

Go To 164

```
      PROGRAM FIVE
      DIMENSION A(3, 3), B(3, 3)
      COMMON A
      READ 10, ((A(I,J), J = 1,3), I = 1,3)
 10   FORMAT (E9.3)
      DO 400 IR = 1, 3
      DO 400 IC = 1, 3
      IF (IR -IC) 100, 200, 300
C     CASE 1: ROW IS SMALLER THAN COLUMN
 100  CALL SUMMA (IR,IC, SUM, IR)
      B(IR, IC) =(A(IR,IC) - SUM)/A(IR,IR)
      GO TO 400
C     CASE 2: ROW EQUALS COLUMN
 200  CALL SUMMA (IR,IR, SUM, IR)
      GO TO 310
C     CASE 3: ROW IS LARGER THAN COLUMN
 300  CALL SUMMA (IC, IC, SUM, IR)
 310  B(IR,IC) = A(IR,IC) - SUM
 400  CONTINUE
      PRINT 500, ((I, J, B(I,J), J = 1,3), I = 1,3)
 500  FORMAT (3(2HB(,I1,1H,I1,3H)Δ=,E9.3))
      END
      SUBROUTINE SUMMA (J,L, SUM, IR)
      DIMENSION A(3, 3)
      COMMON A
      SUM = 0.
      IF (J-1)  10, 10, 20
  10  RETURN
  20  KK = J-1
      DO 30 K = 1, KK
  30  SUM = SUM + A(IR,K) * A(K,L)
      END
      END
```

Go To 165

SIX coming up and FOUR to go
Last lap boy, row, row, row!

The formula for linear coefficient of correlation is:

$$r = \frac{\sum (XY) - N(Mx)\ (My)}{\sqrt{\left[\sum X^2 - N(Mx)^2\right]\left[\sum Y^2 - N(My)^2\right]}}$$

X_i and Y_i are paired members of two groups, X and Y , N is the number of pairs, Mx and My are the MEANS of the respective groups. Summations shown are 1 to N.

Input is from Tape Unit 2, consisting of N paired values of X and Y where each record is made up of X, Y; X, Y; X, Y; X, Y; X, Y; and X, Y (6 pairs). Each X and Y have 4 digits of significance. "N" is defined by the initial control card. A maximum of 10,000 pairs can be analyzed.

Output is on Tape Unit 3. Mx, My, and r are listed under the respective headings shown below: (Headings shown start at left most column of record)

CORRELATION△ANALYSIS

△△MEAN△X△△△△△△△MEAN△Y△△△△△△CORRELATION

Go To 166

```
      PROGRAM SIX
      DIMENSION X(10000), Y(10000)
      READ INPUT TAPE 2, 10, N, (X(I), Y(I), I = 1, N)
10    FORMAT (I5,/,(6E10.4))
      SUMX = 0.
      SUMY = 0.
      SUMXY = 0.
      SUMX2 = 0.
      SUMY2 = 0.
      DO 15 I = 1, N
      SUMX = SUMX + X(I)
      SUMY = SUMY + Y(I)
      SUMXY = SUMXY + X(I) * Y(I)
      SUMX2 = SUMX2 + X(I) ** 2
15    SUMY2 = SUMY2 + Y(I) ** 2
      FN = N
      FMEANX = SUMX/FN
      FMEANY = SUMY/FN
      R = (SUMXY - FN* FMEANX * FMEANY) / (SQRTF((SUMX2 - FN * FMEANX**2)*
     1 (SUMY2 - FN * FMEANY**2)))
      WRITE OUTPUT TAPE 3, 20, FMEANX, FMEANY, R
20    FORMAT (20HCORRELATIONΔANALYSIS,/,2X, 6HMEANΔX, 8X,
     1 6HMEANΔY, 6X, 11HCORRELATION,/, 3(E10.4, 4X))
      END
      END
```

Go To 156

SEVEN coming up, THREE to go
Fire up boy, make it glow!

The "least squares" technique applies to the line, $Y_i = ax_i + b$, involves solving normal equations:

$$\begin{cases} Na + b\sum X_i - \sum Y_i = 0 \\ a\sum X_i + b\sum X_i^2 - \sum X_i Y_i = 0 \end{cases}$$

where N = number of paired points, X_i and Y_i. Summations ($\sum$) shown are from i = 1 to i = N

For example:

GIVEN PAIRED POINTS:

$X_1 = 0.50$, $Y_1 = 0.38$
$X_2 = 1.00$, $Y_2 = 0.82$
$X_3 = 2.50$, $Y_3 = 2.00$

One can see that

$N = 3$, $\sum X_i = 4$, $\sum Y_i = 3.20$
$\sum X_i^2 = 7.50$, $\sum X_i Y_i = 6.01$

NORMAL EQUATIONS are:

$$\begin{cases} 3a + 4b - 3.2 = 0 \\ 4a + 7.5b - 6.01 = 0 \end{cases}$$

One can solve the NORMAL EQUATIONS using Determinants. For example, to solve

$$\begin{cases} ax + by = c \\ dx + ey = f \end{cases}$$

one uses

$$x = \frac{-ce + bf}{ae - bd}$$

$$y = \frac{-af + cd}{ae - bd}$$

X and Y values are READ in by groups. Each group is preceded by a control card indicating the number, N, of paired points in the group. When N = 0, the program is ended. N can vary from 2 to 50, X and Y values contain 3 significant digits. Write a program which will PRINT out "a" and "b" in a form similar to:

A=______ΔΔΔΔΔB=______

Go To 168

```
      PROGRAM SEVEN
      DIMENSION X(50), Y(50)
  10  READ 1, N
   1  FORMAT (I2)
      IF (N) 2, 6, 2
   2  READ 3, (X(I), Y(I), I = 1, N)
   3  FORMAT (2E9.3)
      SUMX = 0.
      SUMY = 0.
      SUMXY = 0.
      SUMX2 = 0.
      DO 4 I = 1, N
      SUMX = SUMX + X(I)
      SUMY = SUMY + Y(I)
      SUMXY = SUMXY + X(I) * Y(I)
  4   SUMX2 = SUMX2 + X(I) **2
      FN = N
      A = (-SUMY * SUMX2 + SUMXY * SUMX)/(FN * SUMX2 - SUMX ** 2)
      B = (-FN * SUMXY + SUMX * SUMY)/(FN * SUMX2 - SUMX ** 2)
      PRINT 5, A, B
  5   FORMAT (2HA=, E9.3, 5X, 2HB=, E9.3)
      Go To 10
  6   END
      END
```

Go To 169

EIGHT coming up, TWO to go
Near the end, nice going Joe!

Simpson's rule for approximating a definite integral is:

$$\int_a^b f(X)dx = \frac{\Delta x}{3}\left(f(a) + 4\,f(a + \Delta x) + 2f(a + 2\Delta x) + 4f(a + 3\Delta x) + \text{---} f(b)\right)$$

For example, using a = 1, b = 2, $\Delta x = 0.25$, and the integral $\int_1^2 \sqrt{1 + 4x\,dx}$, gives:

$f(a) = f(1) = \sqrt{1+4(1)} = \sqrt{5}$
$4f(a + \Delta x) = 4f(1 + .25) = 4f(1.25) = 4\sqrt{1 + 4(1.25)} = 4\sqrt{6}$
$2f(a + 2\Delta x) = 2f(1+2(.25)) \doteq 2f(1.5) = 2\sqrt{1 + 4(1.5)} = 2\sqrt{7}$
$4f(a + 3\Delta x) = 4f(1 + 3(.25)) = 4f(1.75) = 4\sqrt{1 + 4(1.75)} = 4\sqrt{8}$

Since $a + 4\Delta x = 1 + 4(.25) = 2 = b$; the last term is $f(2) = \sqrt{1 + 4(2)} = \sqrt{9}$
Note, the last term is reached when $(a + n\Delta x) = b$ and that no number (2 or 4) appears in front of the first or last terms.

$$\text{So, } \int_1^2 \sqrt{1 + 4x}\,dx = \frac{.25}{3}\left(\sqrt{5} + 4\sqrt{6} + 2\sqrt{7} + 4\sqrt{8} + \sqrt{9}\right) = 2.636 \text{ approx.}$$

Card 1 contains the value Δx, ranging from -.25 to +.25. The first two fields of Card 2 contain "a" and "b" each of which range from -9.99 to + 9.99. Write a program which will use Simpson's rule to approximate and PRINT the results for the integral,

$$\int_a^b \frac{\cos x}{x}\,dx$$

Go To 170

```
      PROGRAM EIGHT
      READ 10, DELTAX, A, B
10    FORMAT (E8.2/2E9.3)
      ENDTERM = COSF (B)/B
      SUM = COSF (A)/A
      J = (B - A)/DELTAX
      C = 4.
      DO 14 N = 1, J
      FN = N
      TERM = C * COSF(A+FN*DELTAX)/(A+FN*DELTAX)
      IF (C - 4.) 13, 12, 13
12    C = 2.
      GO TO 16
13    C = 4.
16    SUM = SUM + TERM
      IF(TERM-ENDTERM) 14, 15, 14
14    CONTINUE
15    SUM = (SUM * DELTAX)/3.
      PRINT 20, SUM
20    FORMAT (4HSUM=, E9.3)
      END
      END
```

Go to 171

NINE coming up, ONE to go
Do this and you're in the dough!

A monthly pay problem involves the following (1) READ IN two cards for each employee-the first is the "employee current pay card", the second is the "employee permanent record" card; (2) Make pay computations, up-date and PUNCH a new set of "employee permanent record" cards; (3) PRINT a "monthly pay record". Assume the following:

CURRENT PAY CARDS FORMAT		PERMANENT RECORD CARDS FORMAT	
COLUMNS	PAY ALLOCATIONS	COLUMNS	PAY ALLOCATIONS
1 - 20	Name of employee	1 - 20	Name of employee
21 - 25	Badge Number	21 - 25	Badge Number
26 - 28	Hours worked (XXX)	26 - 27	No. of dependents
29 - 37	Hourly Rate (X.XX)	28 - 40	Gross pay to date (XXXXX.XX)
38	Problem ends if col. 38 is blank (interpreted as zero)	41 - 53	FICA tax to date (XXX.XX)
		54 - 66	FED. tax to date (XXXX.XX)
		67 - 79	Net pay to date (XXXXX.XX)

The PRINTED record contains:

BADGE__________ (line 1)
MONTHLY△FICA__________ (line 2)
MONTHLY△FED.TAX __________ (line 3)

FICA TAX = 3% of first $4800 earned to date
FED. TAX = .18 [Gross Pay - $54 (No. of dependents.)]

Write the PAY PROBLEM, then

Go To 172

```
    PROGRAM NINE
 5  READ 10, NBADGE1, NHOURS, RATE, ICODE
10  FORMAT (20X, I5, I3, E9.3, I1)
    IF (ICODE) 20, 50, 20
20  READ 25, NBADGE2,NDEPS,GROSS,FICA,FED TAX, PAY
25  FORMAT (20H                    ,I5, I2, 4E13.7)
    IF (NBADGE1-NBADGE2) 26, 30, 26
26  STOP
30  FNHOURS = NHOURS
    GROSSMP = FNHOURS * RATE
    GROSS = GROSS + GROSSMP
    IF (FICA - 144.) 32, 33, 35
32  FICAM = 0.03 * GROSSMP
    IF (FICA + FICAM - 144.) 34, 34, 33
33  FICAM = 144. -FICA
34  FICA = FICA + FICAM
    REFUND = 0.
    GO TO 36
35  REFUND = FICA - 144.
    FICA = 144.
    FICAM = 0.
36  FNDEPS = NDEPS
    FEDTAXM = 0.18* (GROSSMP - 54.*FNDEPS)
    FEDTAX = FEDTAX + FEDTAXM
    PAYM = GROSSMP -FICAM - FEDTAXM + REFUND
    PAY = PAY + PAYM
    PUNCH 25, NBADGE2, NDEPS,GROSS,FICA,FEDTAX,PAY
    PRINT 40, NBADGE2, FICAM, FEDTAXM
40  FORMAT (5HBADGE, I5, /, 12HMONTHLY▲FICA, E11.5,
    /,   15HMONTHLY▲FED.TAX,E12.6)
50  END
    END
```

Go To 173

From: PLATO, THE MODERN PROGRAMMER: (GRIN and Company, 1961)

"If anything can go wrong with a program -- it will."

"Do not believe in miracles -- rely on them."

"Smile -- tomorrow will be worse."

TEN coming up, and then you are done
But three cards are needed for this one!

The Gamma function is defined by the improper integral $\Gamma(x) = \int_0^{\infty} x^{x-1} e^{-x}\, dx$

One technique for evaluating it is the following: (accurate approx. 12 decimal places)

$Y = \Gamma(x) = e^{Z}$ where Z is defined by the series:

Case 1, $x > 6$

$$Z = \left(x - \tfrac{1}{2}\right) \log x - x + \tfrac{1}{2} \log 2\pi + \frac{1}{12x} - \frac{1}{360x^3} + \frac{1}{1260x^5} - \frac{1}{1680x^7} + \frac{1}{1188x^9} - \frac{691}{360360x^{11}}$$

Case 2, $\frac{1}{2} \leq x \leq 6$

First, form D, where $D = x+k$, and k is the smallest integer that makes $x+k > 6$.
Second, form E, where $E = x(x+1)(x+2) \ldots\ldots (D-1)$
Third, find F, where $F = \Gamma(x+k)$ (from case 1)
Fourth, find $\Gamma(x)$, using $\Gamma(x) = F/E$

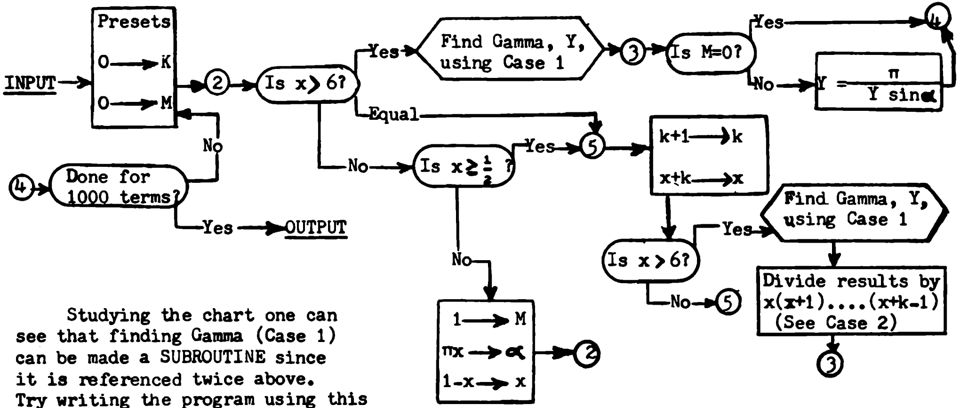
Case 3, $x < \frac{1}{2}$

$$\Gamma(x) = \frac{\pi}{\Gamma(1-x)\sin\alpha}$$

where $\pi x = \alpha$, and $\Gamma(1-x)$ is found by Case 1 or Case 2 above.

Go To 174

Let us assume that 1000 real arguments (one per card), ranging from -5.1 to 15.9 are READ. We want to calculate GAMMA for each of these arguments and print results, 4 per line, separated by 5 spaces each, and accurate to 4 significant figures. Using the previous rules, we first flow chart the problem:



Studying the chart one can see that finding Gamma (Case 1) can be made a SUBROUTINE since it is referenced twice above. Try writing the program using this scheme!

Go To 175

```
      PROGRAM TEN
      DIMENSION X(1000), Y(1000)
      READ 15, (X(I), I = 1, 1000)
   15 FORMAT (E9.3)
      DO 30, I = 1, 1000
      M = 0
      FK = 0.
    2 IF (X(I) - 6.) 7, 5, 6
    7 IF (X(I) -.5) 8, 5, 5
    8 M = 1
      ALPHA = 3.14159 * X(I)
      X(I) = 1. - X(I)
      GO TO 2
    5 FK = FK + 1.
      IF (X(I) + FK - 6.) 5, 5, 9
    9 CALL GAMMA (X(I) + FK, Y(I))
      PRODUCT = 1.
   10 PRODUCT = PRODUCT *(X(I) + FK - 1.)
      FK = FK - 1.
      IF(FK) 12, 12, 10
   12 Y(I) = Y(I)/PRODUCT
      GO TO 3
    6 CALL GAMMA (X(I), Y(I))
    3 IF (M) 20, 30, 20
   20 Y(I) = 3.14159/(Y(I) * SINF(ALPHA))
   30 CONTINUE
      PRINT 60, (Y(I), I = 1, 1000)
   60 FORMAT (4(E10.4,5X))
      END
      SUBROUTINE GAMMA (X, Y)
      Z = (X - .5) * LOGF(X) - X + .5 * LOGF(6.28318)
     1 + 1./(12.*X) - 1./(360.*X**3) + 1./(1260.*
     2 X ** 5) - 1./(1680.*X**7) + 1./(1188.*X**9
     3 ) - 691./(360360. *X**11)
      Y = EXPF (Z)
      END
      END
```

Go to 176

There is a time for everything! A time to live, a time to die,---a time to start and a time to end. You have reached the time to end this course.

If you have doggedly persevered through this course, we congratulate and offer some condolence for having to live with repeated examples, rules, and humor that at times was probably rather "corny".

We started out with <u>one</u> main objective: to teach a <u>minimum</u> FORTRAN course that might enable non-programmers to do <u>most</u> or <u>all</u> of their own programming when the need arises. Again, we reiterate that there has been no attempt to teach <u>every</u> and <u>all</u> rules and concepts of Fortran.

Nevertheless, we are confident that having mastered the "lessons of the cards" you can do a creditable job of programming <u>any</u> problem in Fortran. Only experience through practice can prove whether we are right or wrong.

Why not give it a try?

Go To Hawaii for a rest!

INDEX

INDEX cont.

REFERENCE TABLE OF FORTRAN STATEMENTS

General Form	Definition
A = B	Value of B replaces value of A
CALL Name (a_1, a_2, -- a_n)	Go to subroutine named; args. a_1, a_2, -- a_n
COMMON A, B, etc.	Assigns variables listed to COMMON
CONTINUE	Ends DO, if last loop statement is a transfer
DIMENSION A(n), B(t), etc.	Shows max. range (n, t) of listed variables
DO n I = m_1, m_2, m_3	Loop thru n; I = m_1 to m_2, m_3 = increment
END	Last statement in subprogram or program
FORMAT (specification)	Specifies how I/O data is moved
GO TO n	Go to statement number, n
GO TO (n_1, n_2, -- n_n), i	Go to statement number, n_i
IF (g) n_1, n_2, n_3	Go to n_1, n_2, n_3 as: g <, =, or >0
PAUSE n	Pause, indicate n
PRINT n, LIST	Printer output using FORMAT (n) and LIST
PUNCH n, LIST	Card output using FORMAT (n) and LIST
READ n, LIST	Card input using FORMAT (n) and LIST
READ INPUT TAPE i, n, LIST	Tape i input using FORMAT (n) and LIST
RETURN	Subroutine inner exit
REWIND i	Rewind tape i to start point
STOP n	Stop, indicate n (terminates program)
SUBROUTINE NAME (a_1, a_2, -- a_n)	Identifies subr. and args. a_1, a_2 -- a_n
WRITE OUTPUT TAPE i, n, LIST	Tape i output using FORMAT (n) and LIST

80337